PCT WORLD INTELLECTUAL PROPERTY ORGANIZATION
International Bureau

INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification 6 : G06F 12/00, 13/00 | A1

(11) International Publication Number: WO 97/07461

(43) International Publication Date: 27 February 1997 (27.02.97)

(21) International Application Number: PCT/US96/13238

(22) International Filing Date: 15 August 1996 (15.08.96)

(30) Priority Data:
08/516,293 17 August 1995 (17.08.95) US

(71) Applicant: BORG TECHNOLOGIES, INC. [US/US]; 1341 Cannon Street, Louisville, CO 80027 (US).

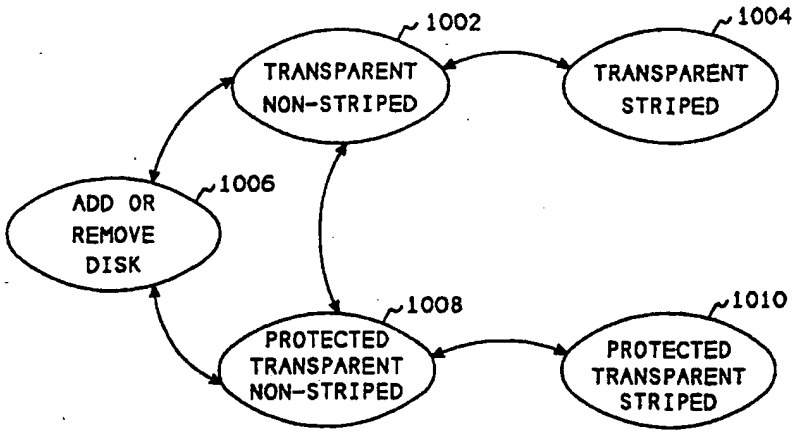
(72) Inventors: STALLMO, David, C.; 59 Beaver Way, Boulder, CO 80304 (US). HALL, Randy, K.; 400 Oneida Street, Boulder, CO 80303 (US).

(74) Agent: YOUNG, James, R.; Suite 385, 12000 N. Washington Street, Denver, CO 80241 (US).

(81) Designated States: CA, European patent (AT, BE, CH, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE).

Published
With international search report.

(54) Title: METHOD AND APPARATUS FOR IMPROVING PERFORMANCE IN A REDUNDANT ARRAY OF INDEPENDENT DISKS



(57) Abstract

A RAID disk array (106, 110, 112, 114, 116) that is adaptable to host I/O traffic, wherein the RAID configuration is hidden from the host computer (102). The system dynamically determines the RAID configuration used to store host data to maximize response time performance and minimize the loss of disk space used for data protection. To maximize response time and avoid a write penalty, small write operations are mapped into RAID 1 configurations, and medium and large write operations are mapped into RAID 3 configurations. These segments are migrated into RAID 5 configurations as a background operation, to minimize the disk space lost. The system hides configuration changes necessary for the addition and/or deletion of disks to the disk array. While these changes are in progress, the disk array (106, 110, 112, 114, 116) remains on-line and all host data is available for access and modification.

FOR THE PURPOSES OF INFORMATION ONLY

Codes used to identify States party to the PCT on the front pages of pamphlets publishing international applications under the PCT.

AM	Armenia	GB	United Kingdom	MW	Malawi
AT	Austria	GE	Georgia	MX	Mexico
AU	Australia	GN	Guinea	NE	Niger
BB	Barbados	GR	Greece	NL	Netherlands
BE	Belgium	HU	Hungary	NO	Norway
BF	Burkina Faso	IE	Ireland	NZ	New Zealand
BG	Bulgaria	IT	Italy	PL	Poland
BJ	Benin	JP	Japan	PT	Portugal
BR	Brazil	KE	Kenya	RO	Romania
BY	Belarus	KG	Kyrgystan	RU	Russian Federation
CA	Canada	KP	Democratic People's Republic of Korea	SD	Sudan
CF	Central African Republic			SE	Sweden
CG	Congo	KR	Republic of Korea	SG	Singapore
CH	Switzerland	KZ	Kazakhstan	SI	Slovenia
CI	Côte d'Ivoire	LI	Liechtenstein	SK	Slovakia
CM	Cameroon	LK	Sri Lanka	SN	Senegal
CN	China	LR	Liberia	SZ	Swaziland
CS	Czechoslovakia	LT	Lithuania	TD	Chad
CZ	Czech Republic	LU	Luxembourg	TG	Togo
DE	Germany	LV	Latvia	TJ	Tajikistan
DK	Denmark	MC	Monaco	TT	Trinidad and Tobago
EE	Estonia	MD	Republic of Moldova	UA	Ukraine
ES	Spain	MG	Madagascar	UG	Uganda
FI	Finland	ML	Mali	US	United States of America
FR	France	MN	Mongolia	UZ	Uzbekistan
GA	Gabon	MR	Mauritania	VN	Viet Nam

METHOD AND APPARATUS FOR IMPROVING PERFORMANCE IN A REDUNDANT ARRAY OF INDEPENDENT DISKS

TECHNICAL FIELD

This invention relates to computer systems and more particularly to disk devices within such computer systems. Even more particularly, the invention relates to a Redundant Array of Independent Disks (RAID) system.

BACKGROUND OF THE INVENTION

In a typical computer system, several disk devices are attached to a host computer. Data blocks are transferred between the host computer and each of the disks as application programs read or write data from or to the disks. This data transfer is accomplished through a data I/O bus that connects the host computer to the disks. One such data I/O bus is called a small computer system interface (SCSI) bus and is commonly used on systems ranging in size from large personal computers to small mainframe computers.

Although each drive attached to the SCSI bus can store large amounts of data, the drives physically cannot locate and retrieve data fast enough to match the speed of a larger host processor, and this limitation creates an I/O bottleneck in the system. To further aggravate the problem, system configurations frequently dedicate one drive to one specific application. For example, in the Unix (tm) Operating System, a Unix file system can be no larger than a single disk, and often a single disk is dedicated to a single file system. To improve performance, a particular file system may be dedicated to each application being run. Thus, each application will access a different disk, improving performance.

Disk arrays, often called redundant arrays of independent disks (RAID), alleviate this I/O bottleneck by distributing the I/O load of a single large drive across multiple smaller drives. The SCSI interface sends commands and data to the RAID system, and a controller within the RAID system receives the commands and data, delegates tasks to independent processes within the array controller, and these independent processes address one or more of the independent disks attached to the RAID system to provide the data transfer requested by the host system.

One way a RAID system can improve performance is by striping data. Striping of data is done by writing data from a single file system across multiple disks. This single file system still appears to the host system as a single disk. since the host system expects a single file system to be

located on a single disk. The RAID system translates the request for data from a single file system and determines which of the physical disks contains the data, then retrieves or writes the data for the host. In this manner, application programs no longer need a file system dedicated to their needs, and can share file systems knowing that the data is actually spread across many different disks.

A stripe of data consists of a row of sectors located in a known position on each disk across the width of the disk array. Stripe depth, or the number of sectors written on a disk before writing starts on the next disk, is defined by the sub-system software. The stripe depth is typically set by the number of blocks that will need to be accessed for each read or write operation. That is, if each read or write operation is anticipated to be three blocks, the stripe depth would be set to three or more blocks, thus, each read or write operation would typically access only a single disk.

Six types of RAID configuration levels have been defined, RAID 0 through RAID 5. This definition of the RAID levels was initially defined by the University of California at Berkeley and later further defined and expanded by an industry organization called the RAID Advisory Board (RAB). Each of the RAID levels have different strengths and weaknesses.

A RAID 0 configuration stripes data across the disk drives, but makes no provision to protect data against loss. In RAID 0, the drives are configured in a simple array and data blocks are striped to the drives according to the defined stripe depth. Data striping allows multiple read and write operations to be executed concurrently, thereby increasing the I/O rate, but RAID 0 provides no data protection in the event one of the disk drives fails. In fact, because the array contains multiple drives, the probability that one of the array drives will fail is higher than the probability of a single drive system failure. Thus, RAID 0 provides high transaction rates and load balancing but does not provide any protection against the loss of a disk and subsequent loss of access to the user data.

A RAID 1 configuration is sometimes called mirroring. In this configuration, data is always written to two different drives, thus the data is duplicated. This protects against loss of data, however, it requires twice as much disk storage space as a RAID 0 system. Thus, RAID 1 provides protection against the loss of a disk, with no loss of write speeds and transaction rates, and a possible improvement in read transaction rates, however RAID 1 uses half the available disk space to provide the protection.

A RAID 2 configuration stripes data across the array of disks, and also generates error correction code information stored on a separate error correction code drive. Usually the ratio of error correction drives to data drives is relatively high, up to approximately 40%. Disk drives ordinarily provide their own redundancy information stored with each block on the drive. Thus,

RAID 2 systems duplicate this redundancy information and require significantly more time and space to be cost effective, so they are seldom used.

A RAID 3 configuration implements a method for securing data by generating and storing parity data, and RAID 3 provides a larger bandwidth for applications that process large files. In a RAID 3 configuration, parity data are stored on a dedicated drive, requiring one drive's worth of data out of the array of drives, in order to store the parity information. Because all parity information is stored on a single drive, this drive becomes the I/O bottleneck, since each write operation must write the data on the data drive and must further update the parity on the parity drive. However, when large blocks of data are written, RAID 3 is an efficient configuration.

RAID 3 provides protection against the loss of a disk with no loss of write or read speeds, but RAID 3 is only suited to large read and write operations. The RAID 3 transaction rate matches that of a single disk and, in a pure implementation, requires the host to read and write in multiples of the number of data disks in the RAID 3 group, starting on the boundary of the number of data disks in the RAID 3 group.

A RAID 4 configuration stores user data by recording parity on a dedicated drive, as in RAID 3, and transfers blocks of data to single disks rather than spreading data blocks across multiple drives. Since this configuration has no significant advantages over RAID 3, it is rarely, if ever, used.

A RAID 5 configuration stripes user data across the array and implements a scheme for storing parity that avoids the I/O bottleneck of RAID 3. Parity data are generated for each write, however, parity sectors are spread evenly, or interleaved, across all drives to prevent an I/O bottleneck at the parity drive. Thus, the RAID 5 configuration uses parity to secure data and makes it possible to reconstruct lost data in the event of a drive failure, while also eliminating the bottleneck of storing parity on a single drive. A RAID 5 configuration is most efficient when writing small blocks of data, such that a block of data will fit on a single drive. However, RAID 5 requires, when writing a block of data, that the old block of data be read, the old parity data be read, new parity be generated by removing the old data and adding the new data. Then the new data and the new parity are written. This requirement to read, regenerate and rewrite parity data is termed a read/modify/write sequence and significantly slows the rate at which data can be written in a RAID 5 configuration. Thus this requirement creates a "write penalty." To minimize the performance impact, RAID 5 stripe depth can be set to be much larger than the expected data transfer size, so that one block of data usually resides on one drive. Consequently, if new data are to be written, only the effected data drive and the drive storing parity data need be accessed to

complete the write operation. Thus, RAID 5 provides protection against the loss of a disk at the cost of one disk's worth of space out of the total number of disks being used; RAID 5 is oriented to transaction processing; and RAID 5 can support large numbers of read operations. However, the read/modify/write sequence causes RAID 5 to have a "write penalty".

In practice, RAID configurations 1, 3, and 5 are most commonly used.

The RAID system manufacturers have had a reasonable understanding of the various tradeoffs for the various RAID levels and have realized that their potential customers will have differing disk I/O needs that would need differing RAID levels. The manufacturers of the first generation of RAID products tended to implement all the levels of RAID (0, 1, 3 and 5) and support the ability of allowing the customer to configure the disks being managed as a disk array to use a mixture of the supported RAID levels.

There are several problems with this approach. The first problem is one of education of the customer. The customer may be an end user, or an integrator, or an original equipment manufacturer (OEM). Providing the customer with the ability to configure the disk array requires that the customer be trained to understand the tradeoffs with the various RAID configurations. The customer also has to be trained to operate a complicated configuration management utility software program.

The main solution to the first problem has been to limit the complexity of configurations, either by the RAID manufacturer who limits the abilities of the configuration management utility program, or by the customer, who chooses a small number of possible combinations for configuration. This solution means that the customer may not necessarily use the best configuration for a given situation, which may lead to disappointing results. Also, the customer may not get full value from the RAID product.

The second problem is that the customer either doesn't know the characteristics of his disk I/O, or these characteristics change over time, or both. Educating the customer and providing a first class configuration management utility program doesn't make any difference if the characteristics of the disk I/O cannot be matched to the best RAID configuration.

The third problem is one of expectations. Customers who buy disks and disk subsystems use two basic measurements to evaluate these systems. The first measurement covers the characteristics of the attached disks. Disks are presently sold as commodities. They all have the same basic features, use the same packaging and support the same standardized protocols. Customers can compare the disks by cost per megabyte, packaging size (5 1/4", 3 ½", etc.), capacity, spin rate and interface transfer rate. These measurements can be used to directly compare

various disk products.

The second measurement is performance when attached to a host computer. It is often possible to use performance tools on the host computer that will report transaction data, such as response time, I/O operations per second, data transfer rate, request lengths in bytes, and request types, such as reads vs writes. It is also common to measure total throughput by using a performance tool to report throughput, or by simply running applications and measuring elapsed time.

A typical customer's expectation is that a new product will not be slower than the products the customer has been using. The customer is happy to get additional protection against the loss of a disk by using a disk array, and is even willing to pay a small premium for this protection, since they can measure the additional cost against the additional protection. But the customer is not generally willing to accept slower performance because of a "write penalty".

Disk array products will continue to be evaluated in the same manner as normal disk products are evaluated. In order for disk arrays to be totally competitive in the disk products market they will have to eliminate the "write Penalty" in all of the commonly used cases.

A fourth problem with requiring the customer to set the configuration is that RAID manufacturers often do not allow dynamic changes to the RAID configuration. Changing the number of disks being used, and changing the levels of protection provided at each target address, often requires that data be migrated to a backup device before the configuration change can be made. After the configuration is changed, the managed disks are re-initialized and the data is then copied back to the disk array from the backup device. This process can take a long time and while it is in progress, the disk array is off-line and the host data is not available.

The current generation of disk arrays appeared in the late 1980's. This generation is divided into completely software versions, that are implemented directly on the host using the host's processor and hardware, and versions using separate hardware to support the RAID software.

The hardware implementation of disk arrays takes multiple forms. The first general form is a PC board that can plug directly into the system bus of the host system. The second general form is a PC board set (one or more boards) that is built into a stand-alone subsystem along with a set of disks. This subsystem often supports some level of fault tolerance and hot plugability of the disks, fans, power supplies and sometimes controller boards.

Generally, the current generation of disk array systems support RAID 5, which requires fairly powerful processors for the level of processing required to support large numbers of RAID 5 requests. The controller board(s) in a disk array, as well as the fault tolerant features, increase the

price of the disk array subsystem. Disk array manufacturers deal with the higher costs in the supporting hardware by supporting large numbers of disks, so that it is easier to amortize the costs of the supporting hardware.

Another problem that disk array manufacturers have is that the capacities of SCSI disks continue to increase rapidly as the cost of the disks continue to decrease rapidly. This trend has resulted in the need to be able to supply disk arrays that have small numbers of disks (3-4) to provide an entry level product, while at the same time, the disk array has to be expandable to allow for growth of the available disk space by the customer. Therefore, disk array controller boards commonly support multiple SCSI channels, typically eight or more, and a SCSI 1 channel can support six or seven disks, reserving one or two ids for initiators, which allows the disk array to support 48 or more disks. This range of disks supported requires controller board(s) that are powerful enough to support a substantial number of disks, 48 or more, while at the same time are cheap enough to be used in a disk array subsystem that only has 3 or 4 disks.

It is thus apparent that there is a need in the art for an improved method and apparatus which allows a dynamic configuration change, allows a disk to be added to the array, or allows a disk to be removed from the array without having to unload and reload the data stored in the array. There is another need in the art for a system that removes the write penalty from a disk array device. The present invention meets these and other needs in the art.

DISCLOSURE OF THE INVENTION

It is an aspect of the present invention to provide a Redundant Array of Independent Disks (RAID) system wherein the particular type of processing being performed is transparent to the host computer system.

It is another aspect of the invention to transpose the data within the RAID system to change from one RAID variation to another.

Another aspect of the invention is to allow a disk to be added to the array, while any data present on the disk, when it is added, remains available.

Yet another aspect is to allow a disk to be removed from the array, while data on all other disks remains available to the host as the disk array re-configures itself to use only the remaining disks.

Still another aspect of the invention is to allow parity protection to be added to or removed from the array.

A still further aspect is to provide a system that usually removes the write penalty while still

providing full RAID functionality.

The above and other aspects of the invention are accomplished in a RAID system that is adaptable to host I/O reads and writes of data. The RAID variations are hidden from the host, thus the system removes the need for a customer to understand the various possible variations within the RAID device. Configuration of the system requires only that the host/customer/system administrator provide a level of configuration that defines the target addresses (such as SCSI ids/LUNs) to which the disk array must respond, the capacity of the defined target addresses, and whether the data at each target address is to be protected against the loss of a disk.

The determination of the RAID variation used to store host data is made dynamically by the disk array of the present invention. This determination is made to maximize response time performance and also to minimize the loss of disk space used for providing protection against the loss of a disk.

The RAID variation can be changed dynamically, on-line, while the data remains available to the host and can be modified by the host. These changes in variation allow the system to reconfigure itself to allow a disk to be deleted from the array, or be added to the array. In addition, a disk being added may have existing data, and this data also remains available to the host and modifiable by the host. After the disk is added, its data will be striped across all the disks of the array.

The system also hides the variation changes necessary for the addition or deletion of disks to the disk array. While these changes are in progress, the disk array remains on-line and all host data is available for access and modification. Additionally, the blocks associated with each target address can have their characteristics changed while the data remains available and modifiable. Thus the host can dynamically add new target address entries, change the number of blocks allocated to the entries, and change the protection afforded to the entries.

To maximize response time, small write operations are written into data blocks organized as a RAID 1 configuration, so there is no write penalty. These RAID 1 blocks are re-written into data blocks organized as a RAID 5 configuration, as a background operation, to minimize the disk space lost.

To maximize response time, medium and large write operations are written into data blocks organized as a RAID 3 configuration, to prevent a write penalty, to maximize bandwidth performance, and to minimize space lost to providing protection.

DESCRIPTION OF THE DRAWINGS

The above and other aspects, features, and advantages of the invention will be better understood by reading the following more particular description of the invention, presented in conjunction with the following drawings, wherein:

Fig. 1 shows a block diagram of a computer system having four data disks managed by a control module of the present invention;

Fig. 2 shows the block diagram of Fig. 1 with an additional parity disk added to the disks being managed by the control module of the present invention;

Fig. 3 shows a block diagram of the hardware of the control module of the present invention;

Fig. 4 shows a diagram illustrating the use of rectangles to manage disk space;

Fig. 5 shows a diagram illustrating the use of squares within rectangles to manage disk space;

Figs. 6-9 show the transparent RAID data organization;

Fig. 10 shows a state diagram of the transitions that are performed by transparent RAID;

Figs. 11-13 show an example of data being transposed from striped to un-striped in transparent RAID;

Figs. 14-16 show examples of data being transposed from striped to un-striped in transparent RAID;

Fig. 17 shows a flowchart of the process of adding a disk to the array;

Fig. 18 shows a flowchart of the process of removing a disk from the array;

Fig. 19 shows the block layout of adaptive RAID;

Fig. 20 shows a flowchart of the process of creating a new block group;

Fig. 21 shows a flowchart of the process of removing a block group;

Fig. 22 shows a flowchart of the adaptive RAID write operation; and

Fig. 23 shows a flowchart of the background processing of adaptive RAID.

BEST MODE FOR CARRYING OUT THE INVENTION

The following description is of the best presently contemplated mode of carrying out the present invention. This description is not to be taken in a limiting sense but is made merely for the purpose of describing the general principles of the invention. The scope of the invention should be determined by referencing the appended claims.

In a typical operating system, such as the Unix(tm) operating system, the attached disks are

independent entities. These disks have mountable file systems defined to use part or all of a disk, however, typically a file system cannot span across more than one disk. Thus a Unix system with 4 disks would have at least four mountable file systems. Normally a single application will use a set of files that all reside on the same file system.

Fig. 1 shows a computer system 100 having a host computer 102 connected to four disks. A host SCSI bus 104 connects the host computer 102 to a control module 106. Those skilled in the art will recognize that any type of I/O bus that connects the host computer 102 to the controller 106 will function with the invention. The control module 106 is connected to four disks 110, 112, 114, and 116 through a SCSI bus 108.

Fig. 1 also shows that the control module 106 is capable of responding to all of the SCSI device ids and logical unit numbers (LUNs) of the managed disks. The control module 106 responds to the set of SCSI ids and LUNs that were originally used for the disks 110, 112, 114, and 116. The SCSI id/LUN that the control module 106 responds to may not have the data that is being requested by the host, however, the host computer 102 will still access the same SCSI ids that were available when the managed disks 110, 112, 114, and 116 were directly connected to the host computer 102. The control module 106 will respond using the same SCSI ids and with the same capacities and characteristics that were available when the managed disks were directly connected to the host.

The original data on the disks is redistributed and evenly striped across the disks being managed by the control module. The effect of this striping is to cause a single application's data to be evenly striped across all of the managed disks.

In an un-striped configuration, the worst case performance occurs when a single application accesses all of its data on a single file system, which is on a single disk. The best case occurs when multiple applications perform a large number of disk requests, resulting in accesses to all the file systems and disks, to provide the best overall throughput.

With the striping provided by the control modules, the worst case performance also occurs when a single application accesses all of its data on a single file system. But since the data is striped across all the disks being managed by the control modules, the accesses will tend to be load balanced across all the disks, so that the worst case operates at the same level as the best case operates in an un-striped configuration. Therefore, the best case, and the worst case performances for the striped data configuration are the same.

When the control module is managing a parity disk, the associated SCSI id/LUN used by the managed parity disk is not available to the host. That is, the host cannot use the parity disk SCSI

id/LUN to communicate with the set of managed disks.

Fig. 2 shows a computer system 200 having five managed disks 210, 212, 214, 216, and 218, wherein the fifth disk 218 is defined as a parity disk. The host computer 202 can use the SCSI ids/LUNs for the first four disks. These SCSI ids/LUNs will show capacities and characteristics of the first four disks 210, 212, 214, and 216 as though these disks were directly attached to the host computer 202.

The user data written by the host computer 202 is striped across all five of the disks, along with the corresponding parity data, to provide protection against the loss of one of the control modules and/or managed disk.

Fig. 3 shows a block diagram of the control module 106. A processor 302 performs the functions of the control module through software, as described below. Input from the host SCSI bus 104 is processed by a SCSI controller 304, and managed disks are controlled through a SCSI controller 308. DMA engines 310 are used for high speed data transfer between the two SCSI busses 104 and 114, and a cache memory 306 is used to buffer data being transferred.

One goal of the system is to allow disks of varying sizes, that is having varying numbers of data blocks, to be managed and to assure that all the blocks on each disk are available to the host computer. When multiple disks are managed, they are organized into multiple "rectangles", where each rectangle has a set of disks that all contain the same number of blocks. The number of rectangles needed is determined by the number of disks that have varying sizes.

Fig. 4 shows an example of four disks, each capable of storing a different number of blocks, and how rectangles would be organized over these disks. Referring to Fig. 4, disk 1 404 is the smallest disk, and it defines the size of rectangle 0. Disk 2 406 is the next largest, and the space remaining on this disk, in excess of the space used in rectangle 0, defines the size of rectangle 1. Similarly, the remaining space on disk 0 402 defines the size of rectangle 2, and the remaining space on disk 3 408 defines the size of rectangle 3.

Because of the number of disks in rectangles 0 and 1, they can be used for all RAID configurations. Rectangle 2 can only be used with RAID 0 and RAID 1, and rectangle 3 can only be used with RAID 0.

Although Fig. 4 shows the rectangles as occupying the same locations on each disk, this is not a requirement. The only requirement is that the amount of space on each disk be the same within a rectangle. The actual location of the space on each disk is not important, so long as it can be readily determined when the disk is accessed.

Another goal of the system is to allow disks that have data already stored on them to be

incorporated into the set of managed disks, and to allow the data from a new disk to be spread across all the managed disks to provide significantly higher levels of performance and to allow protection against the loss of a disk. Still another goal is to dynamically add or remove a disk from the set of managed disks while maintaining the integrity and availability of the data stored in the system.

To accomplish these goals, each rectangle is divided into a set of "squares". A square is a portion of the set of disks contained within a rectangle. The number of blocks in each square is equal to the number of disks in the rectangle multiplied by the depth being used by the rectangle. Each square typically starts at the same logical block number on each disk.

Since the number of blocks in a rectangle is not necessarily an even multiple of the number of blocks in a square, there may be a "partial" at the end of the rectangle, and this partial portion contains the remaining blocks in each disk that cannot fit in a square. These partial blocks do not participate in the striping operation, described below, and thus remain un-striped. They will have data protection, however, since parity can be maintained with an un-striped configuration.

Fig. 5 shows an example of a rectangle and some squares that fit into the rectangle. Referring to Fig. 5, four disks 502, 504, 506, and 508 are shown, wherein the disks comprise a rectangle containing 1000 blocks on each disk. In this example, the depth is four blocks, and since there are four disks, each square contains 16 blocks on each disk. In this example, the rectangle contains 61 squares, and there are six blocks left over on each disk. These left over blocks comprise a partial.

The squares organization is used to allow data to be striped and un-striped across disks. Since each square has the same number of rows and columns, wherein one row is a depth's worth of blocks, and there is one column per disk, matrix transposition is used on a square to stripe and un-stripe data blocks, as will be described below with respect to Figs. 11-16.

The management of data on the disks of the array is layered. At the first level is the management of striping of data blocks and possibly parity blocks. The first level of management is also responsible for sparing and reconstruction operations. This level is called transparent RAID. The second level of management is adaptive RAID, as will be described below.

In transparent RAID, the only configuration information the host/user/system administrator can specify is that an added disk is to be used as a data disk, a parity disk or a spare disk. The disk array uses a disk as a data disk if the type of disk is not defined. The host/user/system administrator can also specify the depth, that is the number of blocks written on a specific disk before writing moves to the next disk.

In transparent RAID, the data blocks on each disk and the parity blocks, if a parity disk is

being used, are automatically striped across all of the managed disks in the set. When a new disk is added to an existing set of managed disks, all the data on the existing disks is re-striped across all the disks including the new disk. The blocks on the new disk are also striped across all of the disks in the managed set.

When a disk is added to the set of the managed disks, the space on this disk is immediately available to the host for all operations. After a disk is added to the set of the managed disks the re-striping of data blocks will commence automatically. During this re-striping operation, the data on the existing disks, as well as the data on the new disk, is available to the host for all operations.

During the re-striping operation the overall performance of the disk array may be reduced because of the disk operations required to re-stripe the data. These disk operations for the re-striping operation are done as background operations giving priority to any normal host I/O requests.

If the disk array is shut down during the re-striping operation, all user data is preserved correctly, and when the disk array is rebooted, the re-striping operation will continue from the point where it stopped.

During the process of re-striping it may be necessary to go through multiple transparent RAID transition variations.

Transparent Variations

Transparent RAID supports a number of variations. The variations are used to allow adding and removing disks to/from a managed set of disks, while remaining on-line to the host and preserving all existing data on the set of managed disks, as well as on the disks being added and/or deleted.

The variations supported are:

transparent non-striped,

transparent striped,

protected transparent non-striped, and

protected transparent striped.

Each of these variations are defined in detail in the following text, including how the transitions between the variations are performed.

Transparent Non-striped

This transparent RAID variation is a direct pass through of data requests, data is not striped, and there is no protection against the loss of a disk, since no parity disk is supported. In essence, this variation treats the disks as totally independent.

Fig. 6 shows four disks 602, 604, 606, and 608 being managed as transparent non-striped, and each disk has its own SCSI id and LUN. Host SCSI requests are passed directly to the managed disks without any id mapping.

Transparent non-striped is one of the base transparent variations used to allow the addition and/or removal of disks. Since there is no striping, the data blocks for each of the data disks are completely contained on the corresponding data disk.

In this variation, when a disk is added to the managed set of disks, it is made immediately available to the host as soon as the disk array completes power up of the disk. In addition, any data that was stored on the added disk is also available to the host.

In this variation the host/user/system administrator can also remove any of the disks from the managed set at any time. Once the disk array is notified to remove a specified disk, the disk array will not respond to any host references to the associated SCSI id and LUN of the removed disk.

Transparent Striped

In this transparent RAID variation, there is no parity data, but the data is striped across all of the managed disks using a depth defined by the host/user/system administrator, or the default depth if none was defined by the host/user/system administrator. To the host, there will still appear to be the same number of SCSI ids that were present when the disks were directly attached, and each of these disks will have the same number of blocks that were available when the disks were directly attached. This supports load balancing of unprotected data.

Fig. 7 shows four disks 702, 704, 706, and 708 in a managed set. The array still responds to SCSI ids 0-3 when the host selects these SCSI ids, but the data is striped across all four of the disks. For example the curved line in each of the disks 702, 704, 706, and 708, represents that the data that was originally stored on the disks is now striped across all the disks.

The rectangles organization, discussed above, is used for all managed disks in all transparent RAID variations, except for transparent non-striped. The rectangles organization is one which will allow all data blocks to be available even when the disks being managed have varying sizes.

The Squares organization, discussed above, is also used for all managed disks for all the variations except transparent non-striped. The Squares organization fits within the rectangles organization, and allows the data in the managed set of disks to be transposed from a non-striped layout to a striped layout, and vice versa, while remaining on-line, and without requiring any disk space to be removed from use by the host/user.

The main feature of the transparent striped variation is that accesses by the host to a single

SCSI id and LUN are distributed across all of the managed disks, thus giving possibly higher levels of throughput and/or response times to the host without making any changes to the host disk driver software.

The main drawback of this variation is that it is not protected and the data for all the managed SCSI ids and LUNs are striped across all disks. Thus a single lost disk will probably effect the users of all SCSI ids, instead of just the users who were specifically placed on the lost disk. Additionally, when the lost disk is replaced it will probably be necessary for the data for all of the SCSI ids, that is all disks, to be restored since all SCSI ids will be missing the data that was on the lost disk.

Protected transparent non-striped

This transparent RAID variation is used to protect a set of managed disks, that do not have the data blocks presently striped, by using a parity disk. This variation is similar to transparent non-striped except that the user blocks are protected against the loss of a disk. This variation appears to the host computer to be the same as the transparent non-striped configuration when the host/user/system administrator wants to add and/or remove one or more disks from the managed set of disks.

Fig. 8 shows four data disks 802, 804, 806, and 808 that are accessible by the host using the associated SCSI id and LUN supported by the disks. The user data is not striped. The fifth disk 810 is a parity disk and contains parity data built from the other four disks. The parity data is completely contained on the parity disk. This parity data is simply the exclusive OR of all the data on disks 802, 804, 806, and 808. done on a byte by byte basis. For example, the first byte of a block of data on disk 802 is exclusive ORed with the first byte of data of a corresponding block on disks 804, 806, and 808, and the exclusive OR result is placed in the first byte of a corresponding block on disk 810. All other bytes of all other blocks are done the same way, such that all the data on disk 810 is the exclusive OR of all the data on the other disks. This parity data can be used to reconstruct the data on any one of the data disks 802, 804, 806, or 808, in the event that a data disk fails. The method of reconstructing this data is well known to those skilled in the art.

Protected Transparent Striped

This transparent RAID variation is the normal transparent RAID variation that is used by adaptive RAID (described below). This mode has completely striped data as well as completely striped parity data across all the disks in the managed set of disks.

Fig. 9 shows four data disks 902, 904, 906, and 908 that are accessible by the host using the associated SCSI id and LUN supported by the disk. The user data is striped. The fifth disk 910 is

defined as the parity disk but it contains striped user data as well as striped parity data.

This is the normal RAID 5 configuration using a set depth that will support the loss of one disk without losing any host data.

Sparing

One or more spares can be specified to be added to support the data in the configuration against loss of one or more disks of user data. When a data disk fails, one of the available spare disks, if there is one available, is automatically chosen and added into the configuration. The blocks in the spare disk are built using data re-generated from the remaining disks in the configuration. While this replacement process is in progress, the configuration has three parts. The first part contains the spare disk with rebuilt data that has replaced the failed disk. The second part contains the blocks that are currently being used to rebuild the data for the spare disk, and this part is locked out to other users while it is being rebuilt. The third part contains the configuration that contains an offline disk, the failed disk, and requires references to data on the off-line disk to be dynamically generated using the other disks.

If a variation transposition to add or delete disks is in progress when a disk fails, the transposition operation will complete the active square being transposed, so the lock around that square can be removed. Then the transposition is suspended until the sparing operation completes. Once the sparing operation is complete, the transposition operation will continue to completion.

When a broken/missing disk is replaced by an operable disk, the new disk will be treated as the spare and be made available for sparing operations.

Depths

The proper depths to be used are dependent upon the characteristics of the data. Shallow depths cause the read and write operations to cross boundaries, thus involving multiple disks in a single transaction. This crossing causes overall throughput in the system to be impacted, since the system will be able to process fewer concurrent requests. A deep depth will reduce the number of boundary crossings but it has several disadvantages. The first disadvantage is that a deep depth will cause reads or writes with high locality to bottleneck on a single disk. The second disadvantage is that a deep depth tends to eliminate the possibility of doing RAID 3 writes or RAID 3 broken reads as effectively as possible.

One way to determine the appropriate depth is to keep a set of heuristics to detect characteristics that can be used to choose a more appropriate depth. The type of heuristic data needed might be:

1) length of requests - if a particular length was predominant, pick a depth that

corresponds well to the request length.

2) boundaries of requests - if the requests are of a particular length, and they fall on particular boundaries, such as multiples of some number, that number can be used for the depth.

3) break statistics into a small number of buckets to allow for more than one set of length and boundaries.

Also, in order to support the squares format, depth must be limited to a reasonable size that will allow the transposition of a square in a short period of time, typically milliseconds or less. Blocks in a square cannot be locked out from a host for a long period of time, such as seconds, or performance may be unacceptable.

To operate efficiently and effectively in accessing, updating, and protecting the host data, the system normally operates in either the transparent striped or protected transparent striped variations. However, before adding or deleting disks, the system must be operating in either the transparent non-striped or protected transparent non-striped variation. Therefore, the system must transit between the different variations.

Fig. 10 shows which transitions between transparent RAID variations can be performed. Referring now to Fig. 10, the transparent non-striped variation 1002 exists when the disks are first placed under management of the array. From this variation, a new disk can be added or removed, as shown by circle 1006. Also from the transparent non-striped variation 1002, the data can be striped over the disks being managed to move to the transparent striped variation 1004.

From the transparent non-striped variation 1002, a parity disk can be added, and parity accumulated, to cause a transition to the protected non-striped variation 1008. From the protected non-striped variation 1008, the data can be striped across the disks for transition to the protected transparent striped variation 1010.

As Fig. 10 shows, the system cannot move directly between the transparent striped variation 1004 and the protected transparent striped variation 1010. If this type of transition is required, the system must move through variations 1002 and 1008 to complete the transition.

Fig. 11 shows a flowchart of the process of transposing the data to accomplish the transitions as described in Fig. 10. Fig. 11 is called whenever the system needs to change the transparent RAID variation. Referring now to Fig. 11, after entry, block 1102 determines if the requested transition is between the transparent non-striped variation and the transparent striped variation. If so, block 1102 transfers to block 1104 which calls the process of Fig. 13 to stripe the data on the disks. After striping all the data, control returns to the caller of Fig. 11. As described

above, data in the last, partial portion of the disk will not be striped.

Block 1106 determines if the transposition is between the transparent striped variation and the transparent non-striped variation. If so, block 1106 transfers to block 1108 which calls the process of Fig. 13 to un-stripe the data on the disks. After un-striping all the data, control returns to the caller of Fig. 11.

Block 1110 determines whether the transposition is between transparent non-striped to protected transparent non-striped. If so, block 1110 goes to block 1112 which exclusive ORs the data within blocks, as described above, to create parity data and store this data on the parity disk. Block 1112 then returns to the caller.

Block 1114 determines whether the transposition is between protected transparent non-striped and protected transparent striped. If so, control goes to block 1116 which calls the process of Fig. 13 to stripe the data across the data disks. Block 1118 then calls the process of Fig. 12 to distribute parity over all the disks. Control then returns to the caller.

If the transposition is from protected transparent striped to protected transparent non-striped, block 1114 goes to block 1120 which calls the process of Fig. 12, once for each square, to combine the parity data onto the parity disk. Block 1122 then calls the process of Fig. 12, once for each square, to unstripe the data. Control then returns to the caller.

Fig. 12 shows a flowchart of the process for distributing or combining parity data over the managed disks. Referring now to Fig. 12, after entry, block 1202 selects the first or next rectangle. Block 1204 then selects the first or next square within the selected rectangle. Block 1206 positions a block position pointer to the first block in the square. All operations of blocks 1212 through 1222 are done relative to the block position pointer.

Block 1212 selects the first, or next, depth group within the square. A depth group is the number of blocks in the depth, over the set of managed disks.

Block 1214 then reads the number of blocks equal to the depth from the disk having the same number as the depth group. For example, if the depth were two, and if the second depth group is being processed, block 1214 would read two blocks from the second disk.

Block 1216 then reads the number of blocks equal to the depth from the parity disk. Block 1218 then writes the parity disk data to the data disk, and block 1220 writes the data disk data to the parity disk. Block 1222 determines if there are more depth groups in the square, and if so, block 1222 returns to block 1212 to process the next depth group.

After all depth groups in the square are processed, block 1222 goes to block 1208 which determines whether there are more squares in the rectangle to process. If there are more squares

to process, block 1208 goes to block 1204 to process the next square.

After all squares in the rectangle are processed, block 1208 goes to block 1210 which determines whether all rectangles within the managed disks have been processed. If there are more rectangles to process, block 1210 goes to block 1202 to process the next rectangle.

After all rectangles have been processed, block 1210 returns to its caller.

Figs. 14 and 15 show an example of the process of combining parity. The process of Fig. 12 is also followed for distributing parity.

Figs. 13A and 13B show a flowchart of the process of striping or un-striping data within the system. Referring now to Fig. 13A, after entry, block 1302 selects the first or next rectangle. Block 1304 then determines if all rectangles have been processed, and returns if they have.

If any rectangles remain, block 1304 goes to block 1306 which selects the first or next square within the rectangle selected in block 1302. Block 1306 determines if all squares within this rectangle have been processed, and if they have, block 1308 goes to block 1302 to get the next square in the selected rectangle.

If all squares have not been processed, block 1301 sets a block position to the beginning of the square. The block position is used in all square processing as the origin of the block, so that all other block selections within the block are relative to the block.

Block 1312 sets the depth group number to zero, and block 1314 selects the first or next data disk starting with data disk zero. Block 1316 skips past a number of blocks to position at the block equal to the depth times the data disk number +1. This block is the first block to be exchanged.

Block 1318 calls Fig. 13B to exchange data at this location, and then block 1320 determines if all the blocks on this data disk, for the entire square, have been processed. If not, block 1320 returns to block 1318 to continue processing this data disk within the square.

After the entire all the blocks on this data disk within the square have been processed, block 1320 goes to block 1322 which increments the data disk number, and also sets the depth group number back to zero. Block 1324 then determines if all data disks within the square have been processed, and if not, returns to block 1316 to process the next data disk within the square.

After all data disks in the square have been processed, block 1324 returns to block 1306 to process the next square.

Fig. 13B shows the process of exchanging data within a square. Referring to Fig. 13B, after entry, block 1350 reads a depth's worth of blocks (i.e. a number of blocks equal to the depth), at the location defined initially by block 1316. Then block 1316 skips past the number of blocks it reads to leave the pointer at the next block after those already read, in preparation for the next pass

through this block.

Block 1352 then reads a depth's worth of blocks from the data disk that has a number equal to the data disk selected in block 1314 plus one plus the depth group number. On this disk, the blocks are read from the location computed by multiplying the disk number (from block 1314) by the depth.

Block 1354 then exchanges these two depth's worth of blocks, and block 1356 increments the depth group number before returning to Fig. 13A.

Figs. 14, 15, and 16 show the data organization of a square of data, and illustrates how this data moves during the transposition between some of the variations. In the example of Figs. 14, 15, and 16, the depth is equal to two, there are four data disks, and one parity disk. Also, in this example, the data blocks are numbered, while the parity blocks for each depth group are represented by the letter "P".

Fig. 14 shows an example of how data is stored in a square within the protected transparent striped variation. Specifically, Fig. 14 illustrates striped data and distributed parity.

Applying the flowchart of Fig. 12 to the data organization of Fig. 14 results in the data organization shown in Fig. 15, which shows striped data and combined parity. In this example, the depth's worth of blocks outlined by the dotted lines 1402 and 1404 are exchanged using the process of Fig. 12. Similarly, the other parity data is exchanged with the non-parity data resulting in the example of Fig. 15, which shows combined parity data within the square.

Applying the flowchart of Figs. 13A and 13B to the data organization of Fig. 15 results in the data organization shown in Fig. 16, which shows un-striped data and combined parity. For example, the depth's worth of blocks outlined by the dotted lines 1502 and 1504 are exchanged, as are the depth's worth of blocks outlined by the dotted lines 1506 and 1508. Similarly, blocks outlined by 1510 and 1512 are exchanged, blocks outlined by 1514 and 1516 are exchanged, the blocks outlined by 1518 and 1520 are exchanged, and the blocks outlined by 1522 and 1524 are exchanged to provide the data organization of Fig. 16, which is non-striped and combined parity.

Add A Disk

When the host/user/system administrator requests that the disk array add one or more disks to the set of managed disks, the system must change the managed disks to a particular transparent variation, as discussed above with respect to Fig. 10. A request to add one or more disks by the host/user/system administrator will be delayed any time there is already a transition operation in progress, or any time there is a sparing operation in progress.

If a disk is to be added while in the protected transparent striped variation, the new disk is

first added to the set of managed disks as a transparent non-striped disk. This makes it immediately accessible to the host, unless it is to be a added as a parity disk. If the disk already contains user data, this data is also immediately available to the host, and the data will be striped along with the other data on the other disks.

Fig. 17 shows a flowchart of the add disk process. Referring to Fig. 17, after entry, block 1702 makes the new disk available to the host computer, as a transparent non-striped disk, if the disk is to be a data disk. Block 1704 then unstripes the existing disks, by calling Fig. 11, to transpose the parity blocks and then transpose the user data blocks for each square on the existing disks. Block 1706 then includes the new disk in the configuration, and block 1708 calls Fig. 11 to transpose the data and parity on the disks, including the new disk, in order to re-stripe the disks.

As the transition proceeds, the variation will be altered to reflect the changes to the data layout on the managed disks. That is, once a square has been transposed, its variation is changed to reflect its new organization, either un-striped or striped, protected or non-protected, depending upon the particular transposition in progress. Thus, during the transition, the system manages the disks as partially striped, partially un-striped, protected or not protected, as the transposition is completed. This allows the data to be available during the transposition, and only the data in a square currently being transposed is not available, and this data is only not available during the short time that the transposition of the square is in progress.

If a shutdown is requested during the transition, the transposition of the active square will complete before the shutdown will be honored.

If the new disk being added is a parity disk, it is not made available to the host, since parity disks are not ordinarily available to the host computer. The system will unstrip the existing disks, and strip the new set of disks and regenerate parity, to include the parity disk.

If the existing disks did not have parity, that is, they were a transparent striped variation, the process proceeds as in Fig. 17, except that there is no parity to transpose.

Remove A Disk

When the host/user/system administrator requests that the disk array remove one or more disks from the set of managed disks, the system must change the managed disks to a particular transparent variation, as discussed above with respect to Fig. 10. A request to remove one or more disks by the host/user/system administrator will be delayed any time there is already a transition operation or sparing operation in progress.

Fig. 18 shows a flowchart of the remove disk process. Referring to Fig. 18, after entry, block 1802 unstripes the existing disks, by calling Fig. 11, to transpose the parity blocks and then

to transpose the user data blocks for each square on the existing disks. Block 1804 then removes the disk from the set of managed disks. Block 1806 then calls Fig. 11 to transpose the data and parity on the remaining disks in order to re-stripe the disks.

As the transition proceeds, the variation will be altered to reflect the changes to the data layout on the managed disks. That is, once a square has been transposed, its variation is changed to reflect its new organization, either un-striped or striped, depending upon the particular transposition in progress. Thus, during the transition, the system manages the disks as partially striped and partially un-striped, as the transposition is completed. This allows the data to be available during the transposition, and only the data in a square being transposed is not available, and this data is only not available during the short time that the transposition of the square is in progress.

If a shutdown is requested during the transition, the transposition of the active square will complete before the shutdown will be honored.

If the new disk being removed is a parity disk, the system will un-stripe the existing disks, and stripe the remaining disks without parity.

If the existing disks did not have parity, that is, they were a transparent striped variation, the process proceeds as in Fig. 18, except that there is no parity to transpose.

Adaptive RAID

The second level of management is called adaptive RAID. Adaptive RAID is built on top of transparent RAID, specifically the protected transparent striped variation.

Adaptive RAID requires configuration information from the host/user/system administrator. Using adaptive RAID, the set of managed disks will appear to the host/user/system administrator as a collection of blocks. The host/user/system administrator defines a set of SCSI ids that have a specified number of blocks associated with each id. The host/user/system administrator no longer has a view into the way the blocks on the managed disks are organized or managed.

Adaptive RAID does not deal with adding and removing disks from the set of managed disks. Instead, when a host/user/system administrator requests that a disk be added or removed from the set of managed disks in the disk array, adaptive RAID is turned off, and the system reverts to the protected transparent striped variation of transparent RAID. Once the transition is made to the protected transparent striped variation, disks can be added and/or removed as defined above.

When using adaptive RAID, a data disk can only be removed if there is enough disk space available, minus the space of the disk being removed. If there is not enough space, the operation will be rejected. Also, a parity disk cannot be removed while adaptive RAID is in use.

In adaptive RAID, each disk is treated as a set of linked groups of blocks. Initially, there is a single group of blocks comprising all the blocks in the disks. This group is called the block pool. The allocation of a block group, defined below, is taken from the block pool.

Figure 19 shows an example of the allocation of blocks. Referring to Fig. 19, three block groups 1902, 1904, and 1906 are shown as linked lists of blocks. A linked list 1908 contains the remaining available blocks, called the block pool. When a read or write request is received, adaptive RAID mapping data structures are used to map the blocks requested by the host into the blocks managed by the transparent RAID. Since all transitions are managed at the transparent RAID level, the adaptive RAID mapping interface to the host interface works regardless of whether the adaptive RAID features are turned on or off.

The structures that support adaptive RAID are always built on a protected transparent striped variation. This variation is the middle ground between the adaptive RAID structures and the variations that allow for disks to be added and removed from the set of managed disks. Any time a disk needs to be added or removed from the set of managed disks, adaptive RAID is turned off and the portions that have been used to expand the configuration beyond transparent RAID will be collapsed back into a normal protected transparent striped variation. While this change is in progress the set of managed disks will remain on-line and accessible by the host. The only effect of turning off the adaptive RAID features is that performance may be impacted because the array will only be supporting normal RAID operations.

Once the additional adaptive RAID portions in the configuration have been collapsed back to a normal protected transparent variation, the striping will be removed by transposing into a protected transparent non-striped variation. After this transposition is complete, the disks are added and/or removed. After all outstanding additions and deletions of disks are completed, the process is reversed, and the disk array will again support the adaptive RAID features.

Transparent RAID allows the management of a disk array to provide load balancing (RAID 0) and/or protected data (RAID 5). Providing adaptive RAID requires configuration information from the host, at a simple level. The host must specify a set of one or more block groups. A user specified block group comprises:

- an id to be used by the host for communicating with the disk array. For SCSI interfaces this is a SCSI id and a LUN.
- the number of blocks to be assigned/allocated to each block group. These blocks are logically numbered from 0 to n-1 where n is the total number of blocks allocated.

- an indication of whether or not the blocks are to be protected.
- an indication of whether or not to initialize the user data blocks to a value of binary zero.

These block groups can be added, deleted or modified at anytime by the host while the disk array is on-line. All existing block groups continue to be on-line and accessible during block group changes.

When a new disk is added to the disk array, the blocks on the added disk are added to the block pool list 1908 within the disk array. As the host defines and adds a new block group, the space for the new block group is taken from the available blocks and reserved for the new block group. The total space specified by the defined block groups includes the parity space needed to provide RAID 5 operations for all protected block groups. The blocks left over from the allocated block groups are used as a block pool to manage adaptive RAID features. Any time the block pool is exhausted, for example because of a high number of host requests, the disk array will revert to transparent RAID operations, so the host must leave an adequate amount of unallocated space for the block pool. The amount of space necessary depends upon the access rate.

Fig. 20 shows a flowchart of the process of creating a new block group. Referring to Fig. 20. after entry, block 2002 receives an id from the host to use for the new block group. Block 2004 receives the number of blocks to allocate to the new block group from the host. Block 2006 removes the number of blocks defined in block 2004 from the block pool and block 2008 connects these blocks to the new block group. Block 2010 then assigns the id received in block 2002 to the new block group, and if initialization has been requested, block 2012 initializes them to binary zero. The host must perform any other desired initialization.

Fig. 21 shows a flowchart of the process of removing a block group. Referring to Fig. 21, when an existing block group is released by the host, block 2102 removes the blocks from the block group, and block 2104 places all the block space removed from the block group into to the block pool. Block 2106 disables the block group id so that the disk array will stop responding to the block group id.

The host specified features of an existing block group can also be changed dynamically. If the size of the block group is increased, the additional blocks are allocated from the block pool and added to the end of the block group's list. The additional blocks will be initialized to zeros, if requested, and the additional blocks will have valid parity if the block group is protected. If the size of the block group is decreased, the specified number of blocks are removed from the end of the block group, and added to the block pool.

The protected state of the block group can be changed, from protected to unprotected or vice versa, in the same manner as transparent RAID. Although this can be a long running operation, depending on the size of the block group, the block group is accessible to other requests while the protected state change is in progress.

Operation of adaptive RAID

The block pool entries are used in to two major ways:

1) When a small write operation is made, a block pool of some minimum size is allocated and given a squares portion that is linked into the appropriate location in the squares portions lists. This block pool entry will be defined using a RAID 1 configuration. This block pool entry will likely be wider than 2 disks. This squares portion is treated specially to allow multiple groups of RAID 1 entries to be created and used.

2) When a larger write operation is made, a block pool entry is allocated and used to provide RAID 3 write operations. The parity data for this block pool entry is not striped, instead, it is always written to the parity disk.

As data areas in normally striped squares portions are replaced by block pool entries, the entire square may be replaced and added to the block pool using a new block pool entry.

The usage of the block pool depends on the write operation being performed:

1) Small random writes (less than one depth's worth) -
These writes are mapped into RAID 1 block pools. This allows the write to be done without a write penalty. These block pool allocations are ultimately written to their original blocks using a RAID 5 write, during background processing.

2) Small sequential writes (less than one depth's worth)-
These writes are mapped into RAID 1 block pools. The block pool allocations are done with extra blocks allocated so that new sequential writes will not immediately require an additional block pool allocation.

3) Medium writes (random or sequential is not important) -
A medium write is one that is large enough to span the disks being managed with a shallow depth. The blocks used are allocated from the block pool and the write operation is performed as a RAID 3 write. Since this is an allocated set of blocks that can start at any logical block, there is never an initial partial square and the ending partial square can have old data, since parity is generated before writing the set of blocks. The trailing partial will be wasted space, since there is no way to

write it later without a write penalty.

4) Large writes (random or sequential is not important) -

A large write is one that is large enough to span all the disks being managed at the depth used in the normal square. This type of write can be done without using the block pool since it can write to the regular square blocks as a RAID 3 write. This type of write can have a partial RAID 3 span in the front and the end. The front partial span is handled as a normal small or medium random write. The trailing partial RAID 3 span is also treated as a small or medium random write.

Fig. 22 shows a flowchart of the adaptive RAID write operation. Referring to Fig. 22, when a write command is received, block 2202 determines if the size of the data being written is less than the size of the depth. That is, will the write be contained on a single disk. If so, block 2202 transfers to block 2204 which determines whether this write sequentially follows the last write. If the write is not sequential, block 2204 goes to block 2206 which allocates new space for the data from the block pool. The amount of space allocated is two times the size of the data being written, since the write will be performed as a RAID 1 write, which mirrors the data. After defining the size to be allocated, block 2206 goes to block 1121 which allocates the space from the block pool, block 2214 then assigns this space to a RAID 1 configuration, and block 2216 writes the data.

If the write sequentially followed the last write, block 2204 goes to block 2208 which determines whether space remains in the space allocated to the last write to contain this write. If so, block 2208 goes to block 2216 to write the data in the previously allocated space from the block pool.

If no space is available, block 2208 goes to block 2210 which defines the space as two times the data size, plus extra space to accommodate additional sequential writes. The amount of extra space allocated varies with the number of sequential writes that have been performed recently.

After defining the space, block 2210 goes to block 2212 to allocate the space, then block 2214 assigns RAID 1 configuration to the space, and block 2216 stores the data.

If the data size is larger than the depth, block 2202 goes to block 2218 which determines whether the data size will span all the disks, that is, is the size large enough for a RAID 3 write. If the data will span all disks, block 2218 goes to block 2226, which writes the data directly to a square, since the write can be performed as a RAID 3 write, with no write penalty.

If the data size is larger than one disk, but smaller than the span of all disks, block 2218 goes to block 2220 which allocates data space for the write from the block pool. This data space is the size of the data being written, plus parity. Block 2222 then assigns this space as RAID 3

configuration, and block 2224 writes the data to the space.

Aging/Recollection Considerations

When a block pool entry is allocated, it uses up a limited resource (i.e. the blocks in the block pool). At some point it may be necessary to move the data being stored in these blocks back to their original blocks.

There are a number of considerations for this decision:

1) When block pool allocations are made for a RAID 1 operation, unused blocks are left in the original portion of the data square, which is inefficient. The allocated block pool space is also inefficient, since half of the disk blocks are used for parity, whereas storing the data back into the square, in a RAID 5 layout, uses less than half the blocks used for parity. If the RAID 1 blocks are updated frequently by the host, however, it is advantageous to leave the blocks allocated in the block pool, to avoid the overhead of constantly cleaning up and then reallocating the block pool entries.

2) When block pool allocations are made for a RAID 3 write, unused blocks are left in the original portion of the data square, which is inefficient. The allocated block pool space is efficient, however, since it is stored in a RAID 3 configuration. If entire rows are replaced, the blocks in the original portion can be given to the block pool.

3) Block pool allocations in RAID 1 configuration are always returned to their original block locations, to free up the block pool area for other uses.

4) Depth considerations determine when and if to move RAID 3 block pool allocations back to their original locations. When a write occurs, space may be allocated at a depth less than the depth of the data in the squares, to allow a smaller write to become a RAID 3 write. In this case, the data will be moved back to the squares where it is stored more efficiently.

5) The more block pool allocations there are, the larger the configuration data structures, used to manage the block pool, become. This growth can result in longer search times and ultimately in running out of space for the configuration data structures. Therefore, the system constantly works in the background to collapse the configuration data structures back to their original rectangles configuration. The main reason to not continually collapse the configuration is because "hot spots", wherein the host updates an area of data frequently, should be left in a RAID 1

configuration.

6) When blocks are allocated for a RAID 1 allocation of a small write, extra blocks are allocated. These extra blocks are used to allow sequential small writes to use the extra blocks without additional non-consecutive allocations. These extra blocks are managed such that if the block pool is exhausted the extra blocks that are not being used can be removed and returned to the available block pool to be used for other allocations.

7) Block pool space has a different, more shallow, depth for RAID 3 allocations to ensure that less space is wasted. In this case, the system may end up with more operations where subsequent read operations cross depth boundaries and cause a lower throughput.

Fig. 23 shows a flowchart of the background processing described above. Referring to Fig. 23, after entry, block 2302 determines whether any block pool allocations have been made. If not, or after processing all of them, block 2302 returns. If unprocessed block pool allocations remain, block 2302 goes to block 2304 which determines whether any RAID 1 configuration allocations are present. If so, block 2304 transfers to block 2306 which selects the first or next RAID 1 allocation. Block 2308 determines whether all RAID 1 allocations have been processed, and if not, goes to block 2310 which determines whether the RAID 1 allocation selected in block 2306 has been recently updated. If a block pool allocation has been recently updated, it will not be moved back to the squares, since it is more efficient to keep it as a RAID 1 allocation, rather that frequently re-allocating new block pool space. Although how often updates must occur to prevent rewriting back into the squares space is dependent upon the type of activity from the host, one example might be to re-write after no updates have occurred within the last second. Therefore, if the block pool allocation has been recently updated, block 2310 goes back to block 2306 to select the next block pool allocation.

If the allocation has not been recently updated, block 2310 goes to block 2312 which writes the data from the block pool allocation back into the location in the square, and block 2314 frees the space from the block pool allocation and returns it to the block pool. Block 2314 then returns to block 2306 to process the next RAID 1 block pool allocation.

After all RAID 1 block pool allocations have been processed, or if there are no RAID 1 block pool allocations, control goes to block 2316 to process RAID 3 allocations. Block 2316 determines if there are RAID 3 allocations to process, and if so, goes to block 2318 which selects the first or next RAID 3 allocation. Block 2320 then determines if this allocation has an inefficient

depth, as discussed above. If so, block 2320 goes to block 2322 which writes the data back to the original squares, and then block 2324 frees the block pool allocation space and returns it to the block pool. Block 2324 then returns to block 2316 to process the next RAID 3 allocation.

If the depth is efficient, block 2320 goes to block 2326 which frees the space in the original square to the block pool, and connects the block pool allocation space, containing the RAID 3 data, into the location of the original square. Thus the data is connected into the original square without being moved. Block 2326 then returns to block 2316 to process the next RAID 3 allocation.

After all RAID 3 allocations have been processed, block 2316 returns to block 2302.

Request Processing

Adaptive RAID can easily end up with a substantial number of squares portions. These squares portions are independent and may contain data in a variety of RAID configurations. This complexity leads to several requirements and/or implementations:

1) The searching of the configuration can be linear when the configuration is small. But when the configuration gets large it can require substantial time to do linear searching. Thus it is necessary to provide additional support using hardware and/or software to limit the time spent searching the configuration data;

2) Because of the dynamic nature of the configuration, all read and write operations must lock sector ranges to assure that concurrent requests cannot cause changes to the same location.

3) Access to the configuration structures must be tightly limited to as few procedures as possible to assure integrity of the structure, thus only one process/request can be accessing and/or modifying the configuration structures at any one time. A read/write request will result in a list to be generated for the physical sectors involved. This list can only be generated after the sector range lock is executed. Once the list is generated, the configuration structures are not used, so they may be modified by other requests. The sector range lock assures that the physical sectors specified in the list cannot change position or be moved in the configuration.

4) The configuration structure can be very dynamic, it must be saved across power off situations, and it must be able to survive failures of the controller as well as short power failures.

Having thus described a presently preferred embodiment of the present invention, it will be understood by those skilled in the art that many changes in construction and circuitry and widely differing embodiments and applications of the invention will suggest themselves without departing

from the scope of the present invention as defined in the claims. The disclosures and the description herein are intended to be illustrative and are not in any sense limiting of the invention, defined in scope by the following claims.

CLAIMS

What is claimed is:

1. In an array of storage devices (106, 110, 112, 114, 116) accessible from a host computer system (102), wherein data stored by said host computer system (102) on each one of said storage devices (106, 110, 112, 114, 116) is distributed by said array across all storage devices in said array, a method of organizing storage devices within said array, said method comprising the steps of:

(a) organizing data blocks on said storage devices into at least one block group (1902, 1904, 1906) and a block pool (1908);

(b) when data is written to said array, allocating blocks from said block pool (1908) to form a group of allocated blocks to store said data;

(c) when data is not being written to said array, placing data from said allocated blocks into said at least one block group (1902, 1904, 1906).

2. The method of claim 1 wherein step (c) comprises the step of copying said data to said at least one block group (1902, 1904, 1906).

3. The method of claim 1 wherein step (c) comprises the step of exchanging blocks from said at least one block group (1902, 1904, 1906) with said allocated blocks.

4. The method of claim 1 wherein a size of each of said at least one block group (1902, 1904, 1906) is dynamically changeable by a user of said method.

5. The method of claim 1 wherein a protection status of each of said at least one block group (1902, 1904, 1906) is dynamically changeable by a user of said method.

6. The method of claim 1 wherein data stored in said at least one block group (1902, 1904, 1906) is stored in a first configuration and data stored in said allocated blocks is stored in a second configuration.

7. The method of claim 6 wherein said first configuration and said second configuration are selected by said array.

8. The method of claim 1 wherein step (b) further comprises the steps of:

(b1) when said data being written is not located sequentially after data previously written, allocating more blocks than necessary to contain said data being written; and

(b1) when said data being written is located sequentially after data previously written, writing said data into allocated blocks containing said previously written data.

9. The method of claim 1 wherein step (b) further comprises the steps of:

(b1) when said data being written is no larger than an amount of data that can be stored on a single disk within said array, allocating an amount of said blocks sufficient to allow said data to be written twice, wherein said data is duplicated within said allocated blocks.

10. The method of claim 1 wherein step (b) further comprises the steps of:

(b1) when said data being written is an amount of data that requires writing to all disks of said array, writing said data in said at least one block group (1902, 1904, 1906), wherein no blocks are allocated from said block pool.

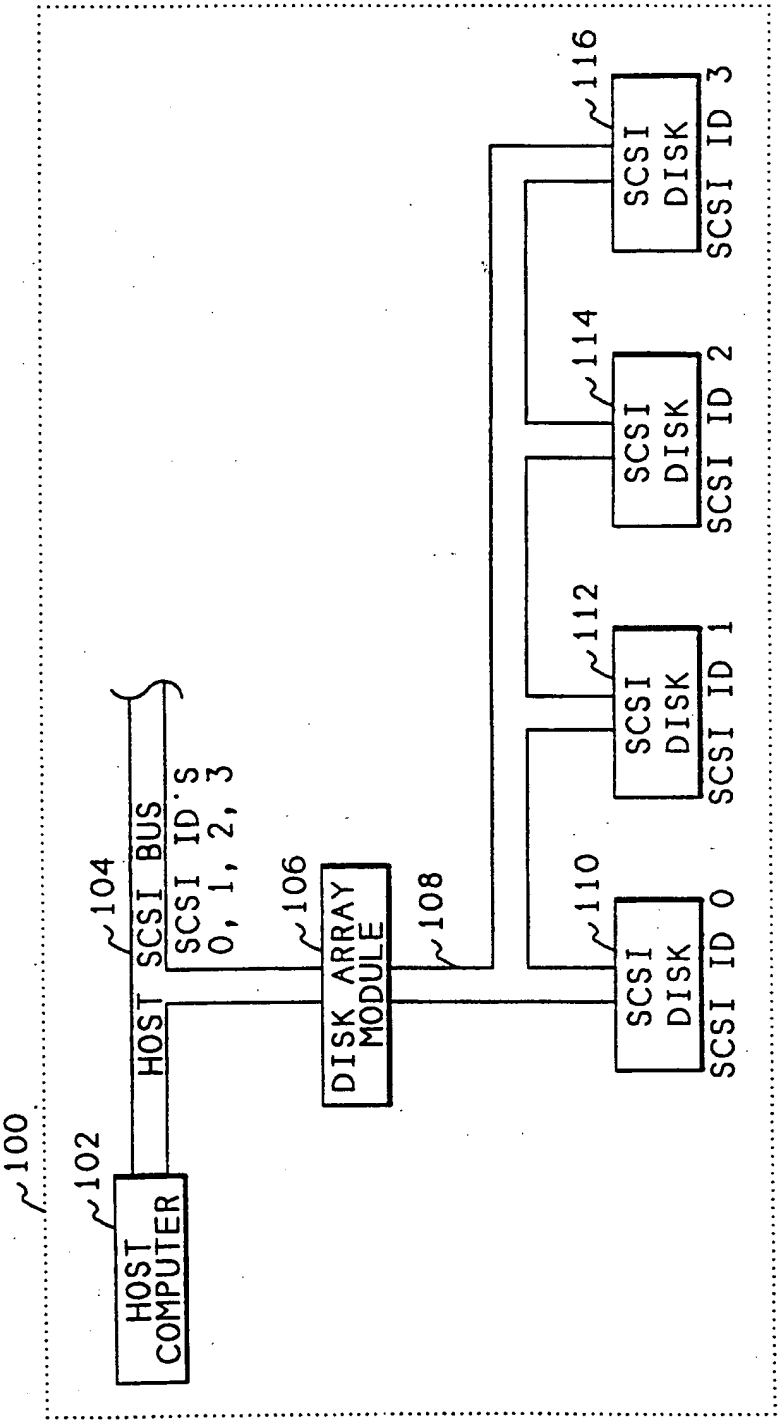

100
102
HOST COMPUTER
104
HOST SCSI BUS
SCSI ID'S 0, 1, 2, 3
106
DISK ARRAY MODULE
108
110
SCSI DISK
SCSI ID 0
112
SCSI DISK
SCSI ID 1
114
SCSI DISK
SCSI ID 2
116
SCSI DISK
SCSI ID 3

FIG. 1

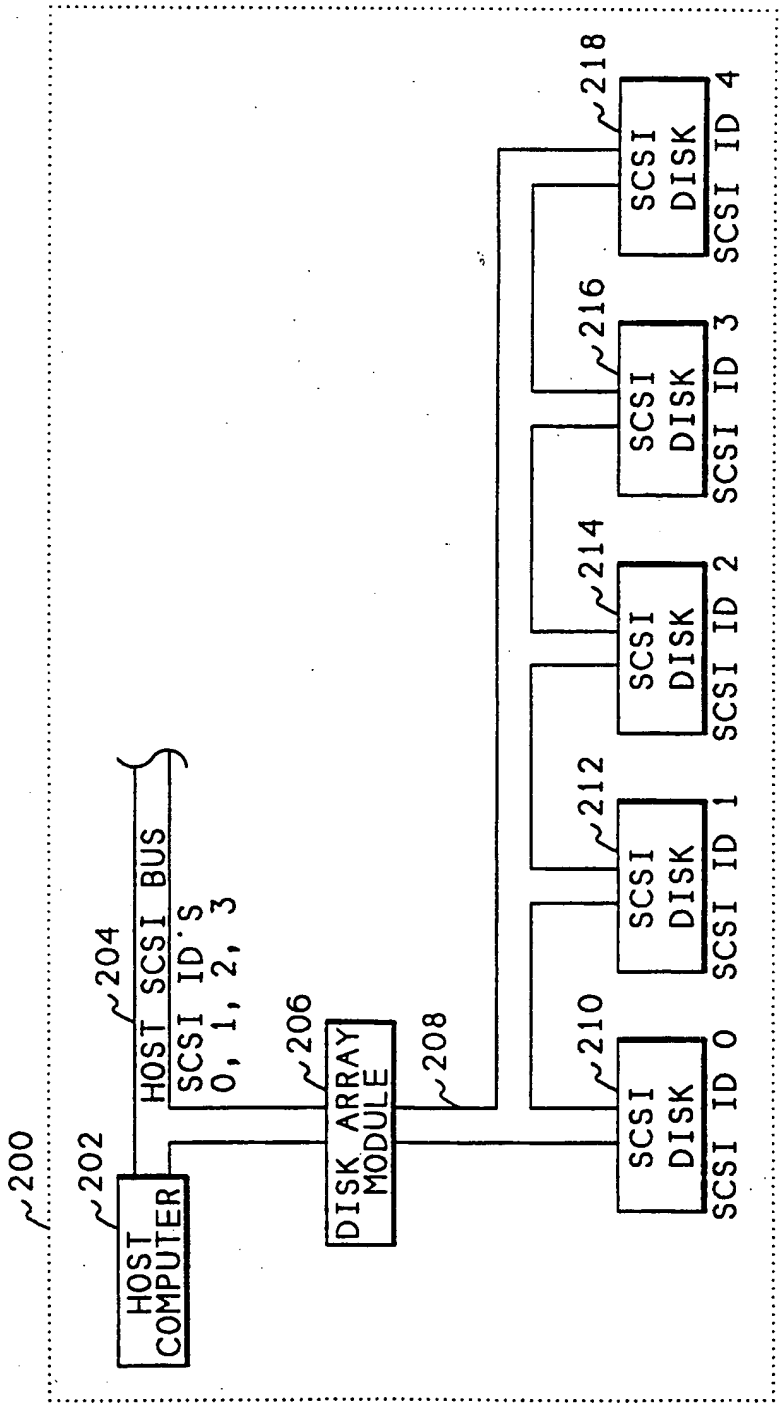

200
202
HOST COMPUTER
204
HOST SCSI BUS
SCSI ID'S 0, 1, 2, 3
206
DISK ARRAY MODULE
208
210
SCSI DISK
SCSI ID 0
212
SCSI DISK
SCSI ID 1
214
SCSI DISK
SCSI ID 2
216
SCSI DISK
SCSI ID 3
218
SCSI DISK
SCSI ID 4

FIG. 2

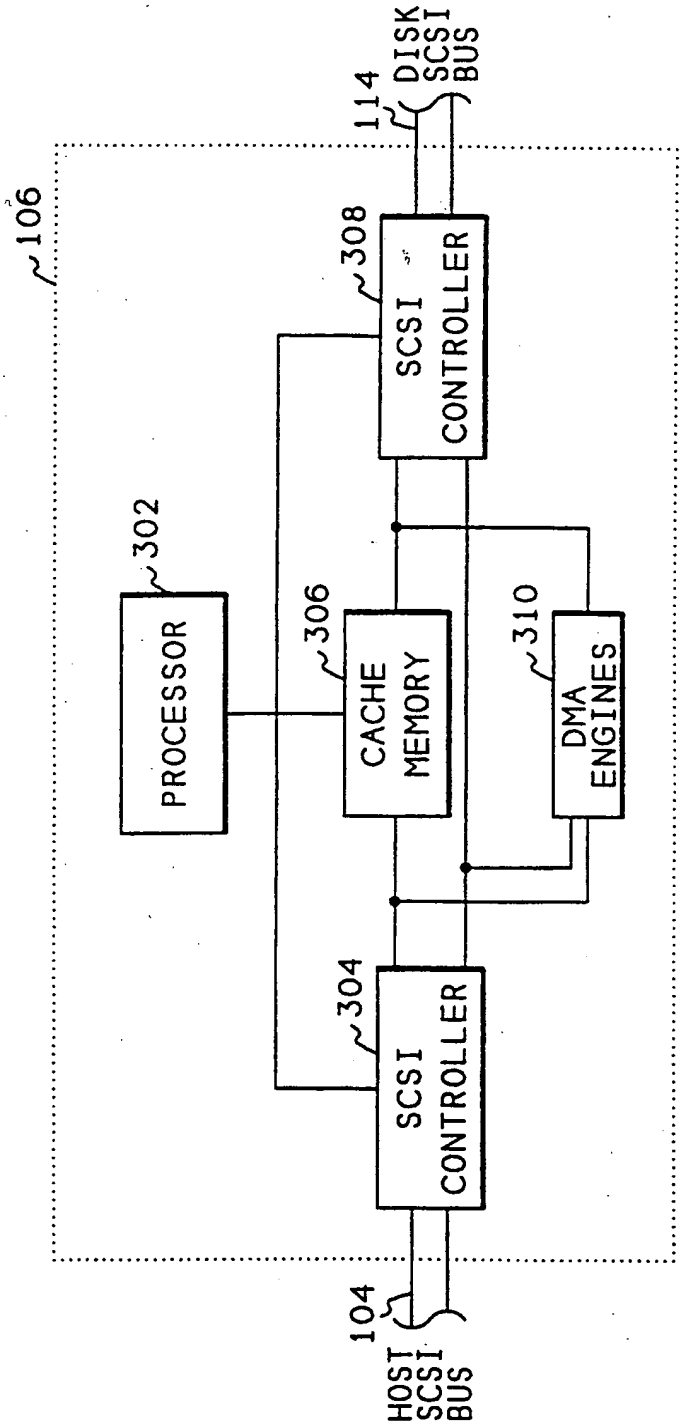

106
114
DISK SCSI BUS
308
SCSI CONTROLLER
302
PROCESSOR
306
CACHE MEMORY
310
DMA ENGINES
304
SCSI CONTROLLER
104
HOST SCSI BUS

FIG. 3

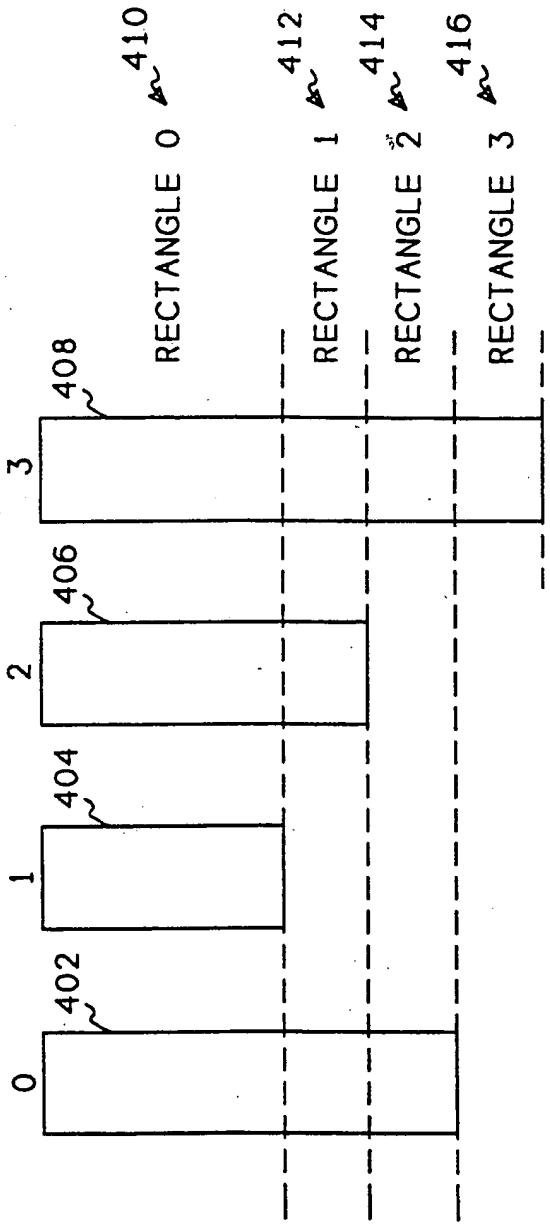

RECTANGLE 0
RECTANGLE 1
RECTANGLE 2
RECTANGLE 3
410
412
414
416
408
406
404
402
3
2
1
0

FIG. 4

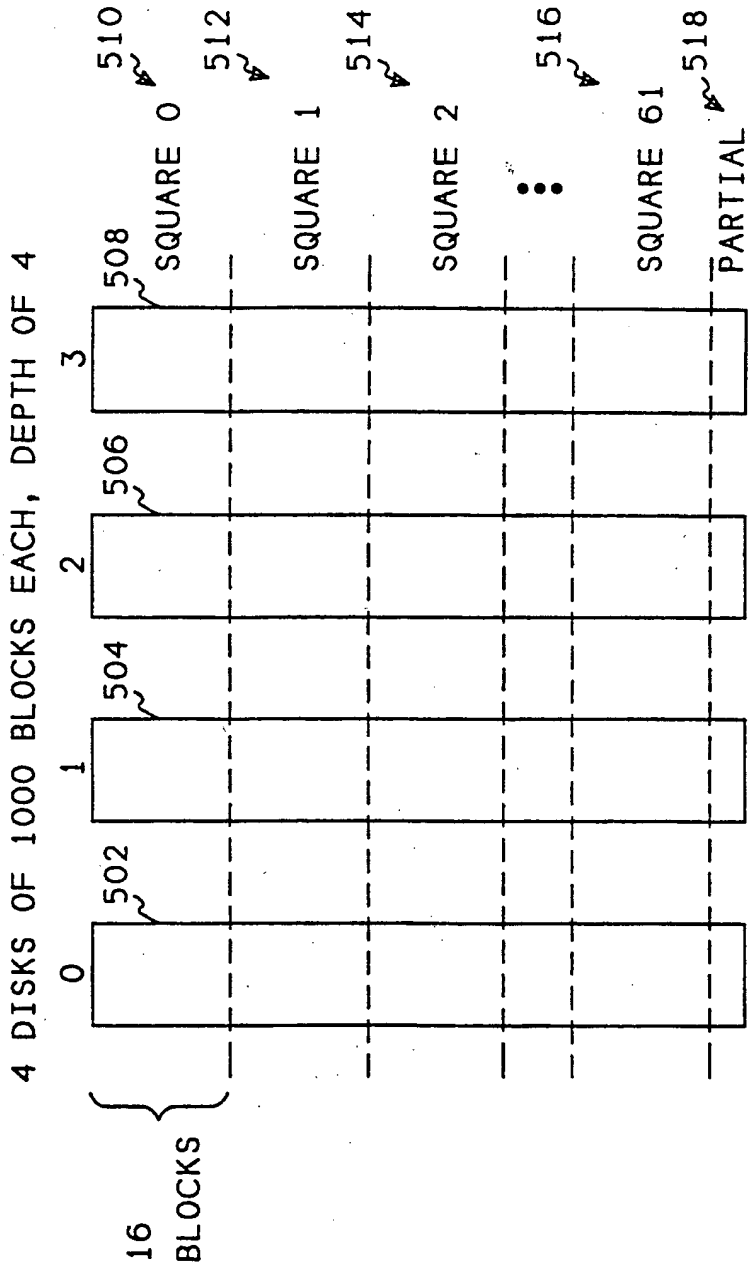

4 DISKS OF 1000 BLOCKS EACH, DEPTH OF 4
16 BLOCKS
0
1
2
3
502
504
506
508
510
512
514
516
518
SQUARE 0
SQUARE 1
SQUARE 2
SQUARE 61
PARTIAL

FIG. 5

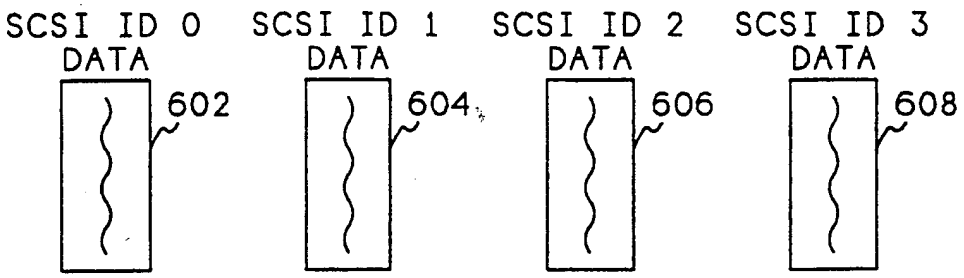

SCSI ID 0
DATA
602
SCSI ID 1
DATA
604
SCSI ID 2
DATA
606
SCSI ID 3
DATA
608

FIG. 6

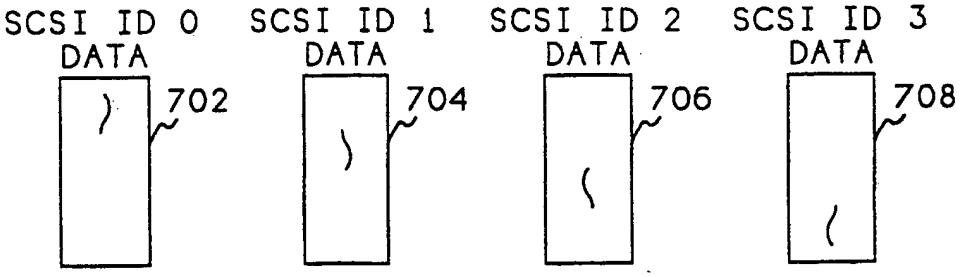

SCSI ID 0
DATA
702
SCSI ID 1
DATA
704
SCSI ID 2
DATA
706
SCSI ID 3
DATA
708

FIG. 7

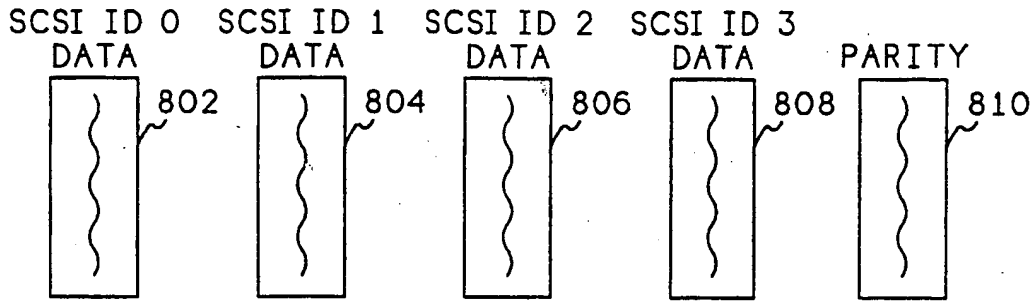

SCSI ID 0
DATA
802
SCSI ID 1
DATA
804
SCSI ID 2
DATA
806
SCSI ID 3
DATA
808
PARITY
810

FIG. 8

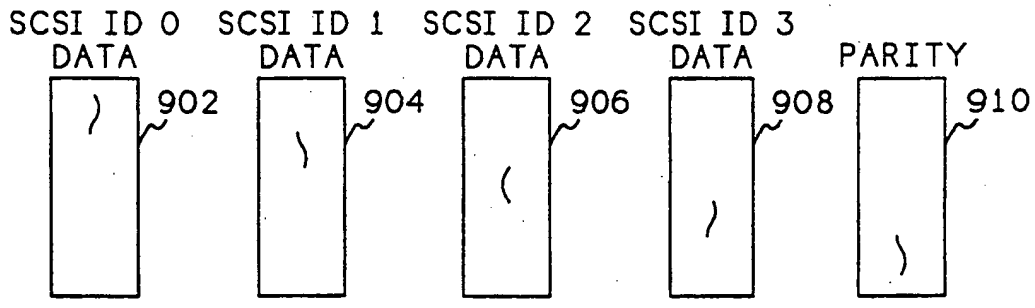

SCSI ID 0
DATA
902
SCSI ID 1
DATA
904
SCSI ID 2
DATA
906
SCSI ID 3
DATA
908
PARITY
910

FIG. 9

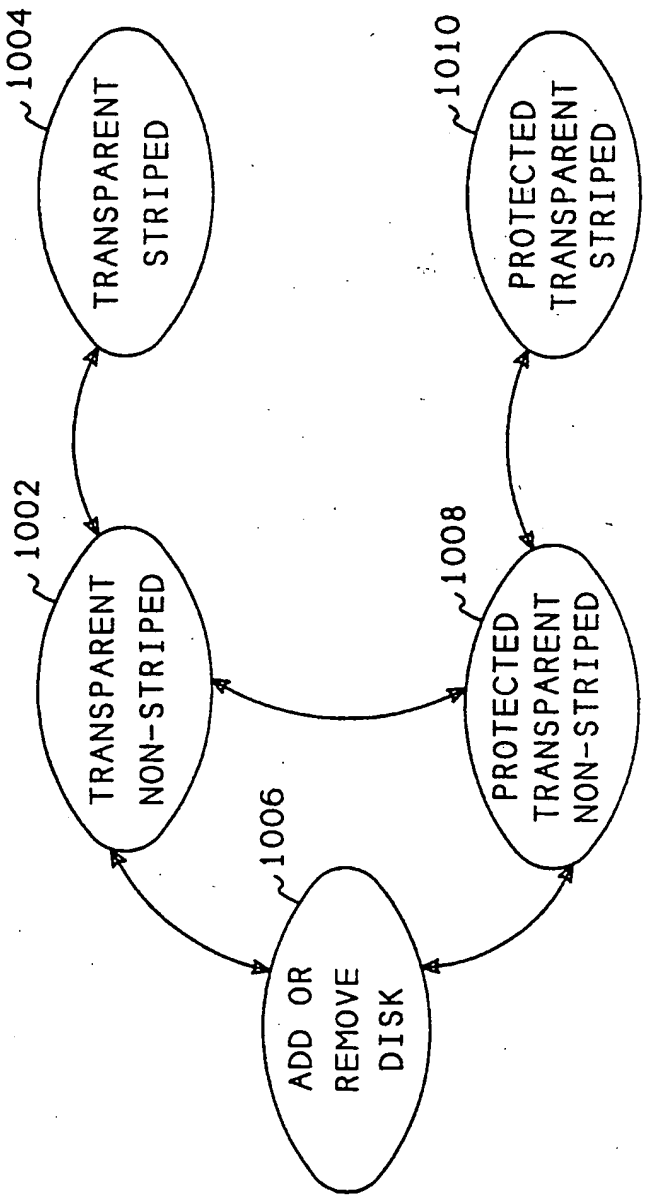

1004
TRANSPARENT STRIPED
1010
PROTECTED TRANSPARENT STRIPED
1002
TRANSPARENT NON-STRIPED
1008
PROTECTED TRANSPARENT NON-STRIPED
1006
ADD OR REMOVE DISK

FIG. 10

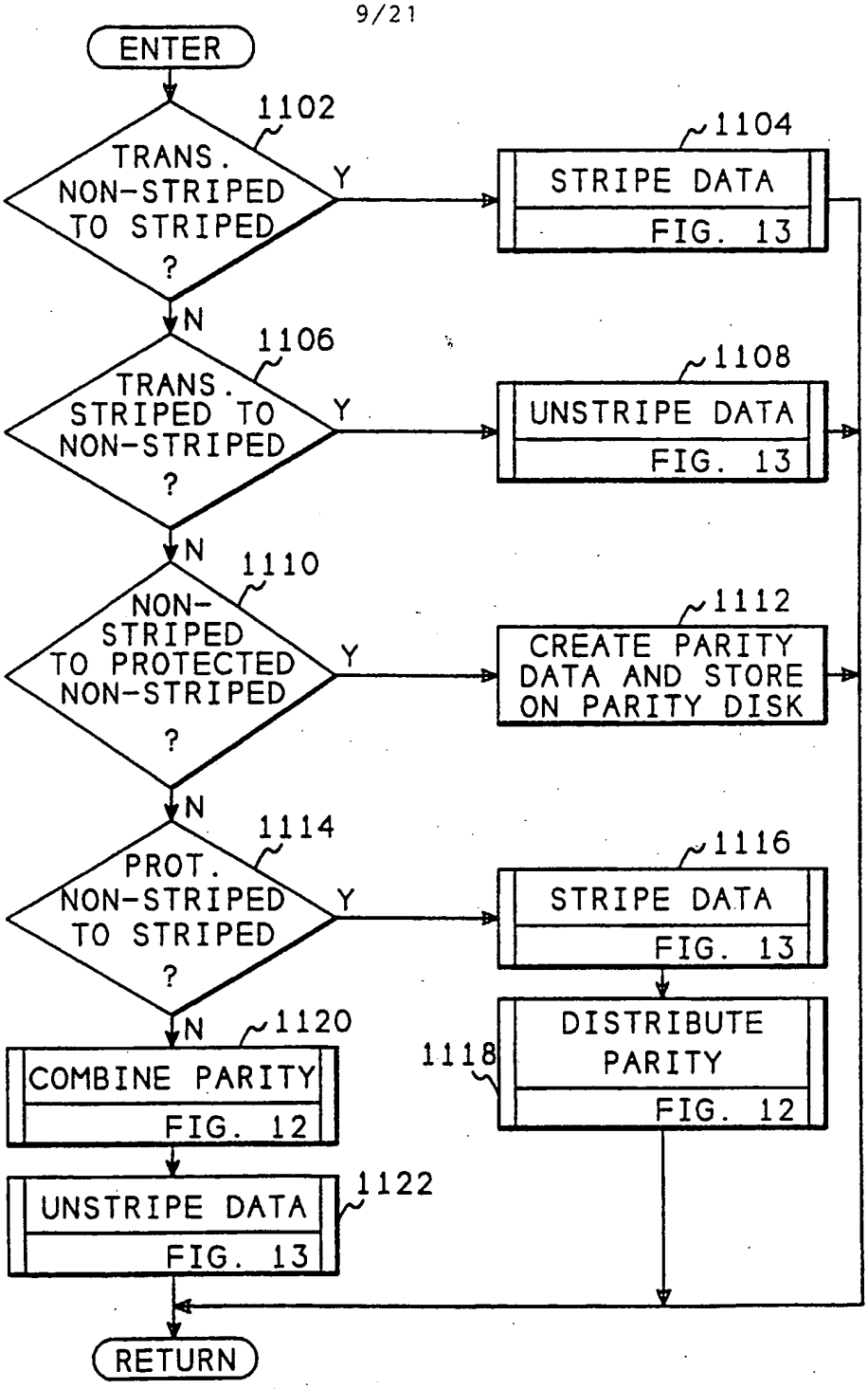

ENTER
1102
TRANS. NON-STRIPED TO STRIPED ?
Y
1104
STRIPE DATA
FIG. 13
N
1106
TRANS. STRIPED TO NON-STRIPED ?
Y
1108
UNSTRIPE DATA
FIG. 13
N
1110
NON-STRIPED TO PROTECTED NON-STRIPED ?
Y
1112
CREATE PARITY DATA AND STORE ON PARITY DISK
N
1114
PROT. NON-STRIPED TO STRIPED ?
Y
1116
STRIPE DATA
FIG. 13
1118
DISTRIBUTE PARITY
FIG. 12
N
1120
COMBINE PARITY
FIG. 12
1122
UNSTRIPE DATA
FIG. 13
RETURN

FIG. 11

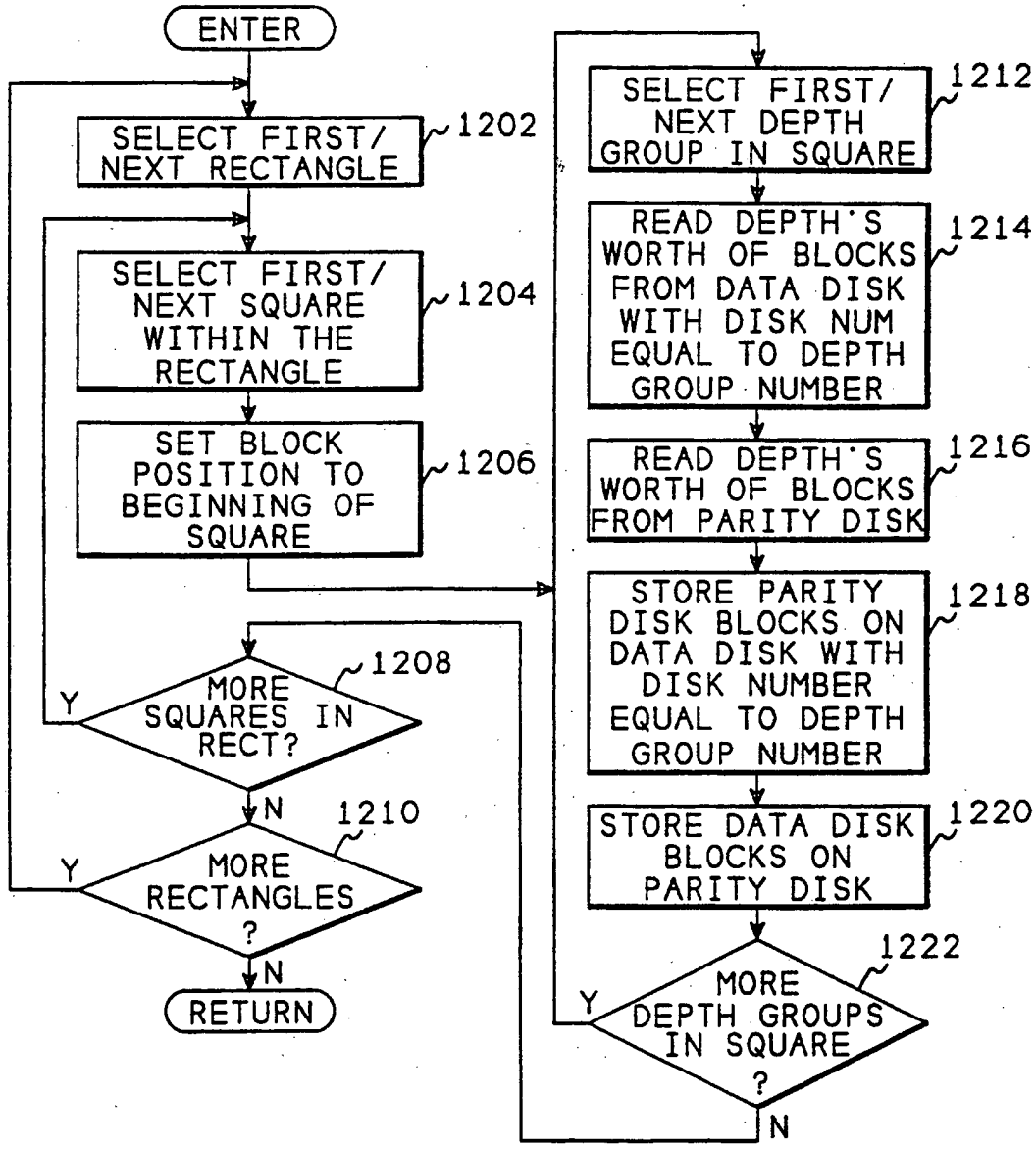

ENTER
SELECT FIRST/ NEXT RECTANGLE
1202
SELECT FIRST/ NEXT SQUARE WITHIN THE RECTANGLE
1204
SET BLOCK POSITION TO BEGINNING OF SQUARE
1206
MORE SQUARES IN RECT?
1208
Y
N
MORE RECTANGLES ?
1210
Y
N
RETURN
SELECT FIRST/ NEXT DEPTH GROUP IN SQUARE
1212
READ DEPTH'S WORTH OF BLOCKS FROM DATA DISK WITH DISK NUM EQUAL TO DEPTH GROUP NUMBER
1214
READ DEPTH'S WORTH OF BLOCKS FROM PARITY DISK
1216
STORE PARITY DISK BLOCKS ON DATA DISK WITH DISK NUMBER EQUAL TO DEPTH GROUP NUMBER
1218
STORE DATA DISK BLOCKS ON PARITY DISK
1220
MORE DEPTH GROUPS IN SQUARE ?
1222
Y
N

FIG. 12

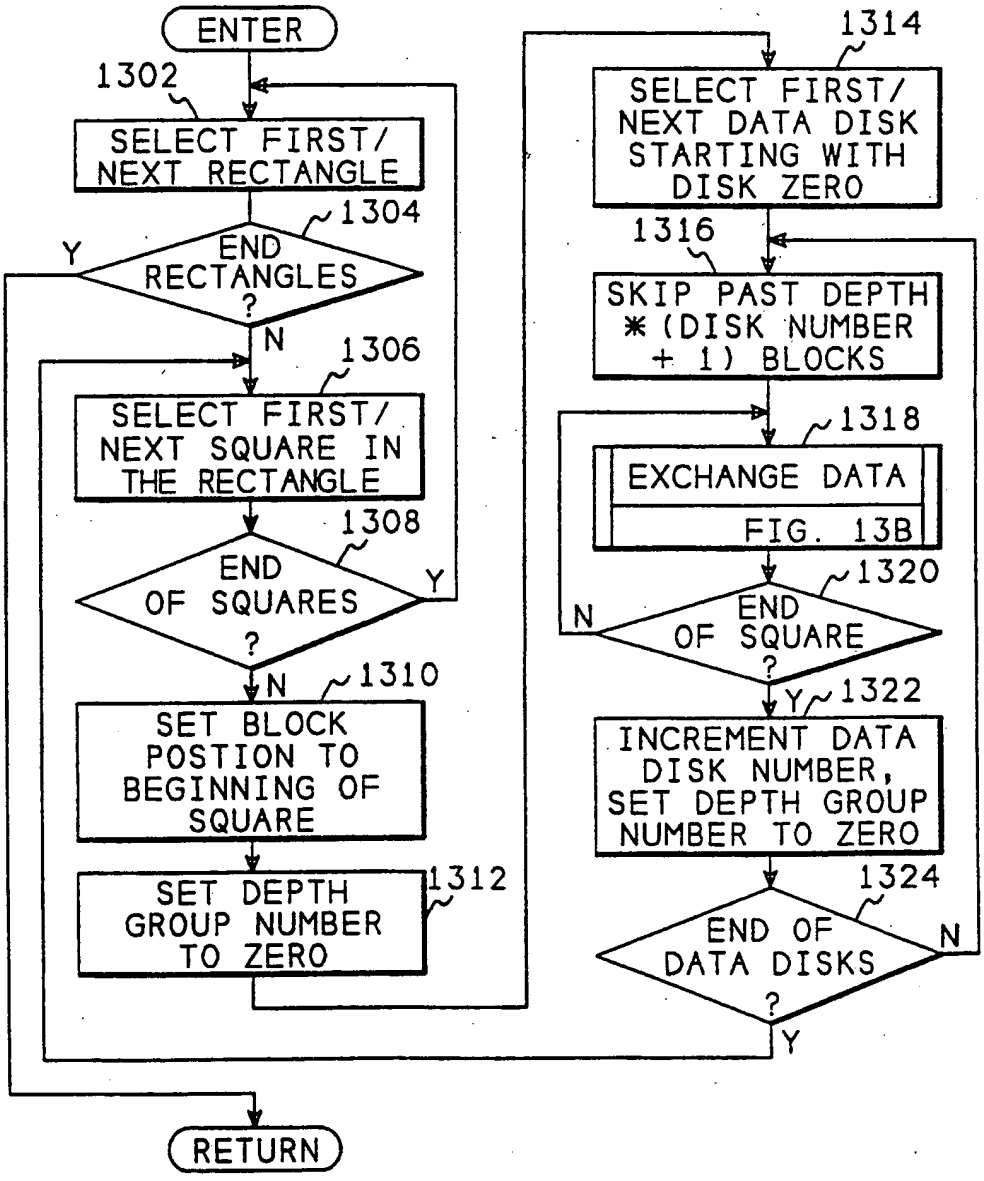

ENTER
1302
SELECT FIRST/ NEXT RECTANGLE
1304
END RECTANGLES ?
Y
N
1306
SELECT FIRST/ NEXT SQUARE IN THE RECTANGLE
1308
END OF SQUARES ?
Y
N
1310
SET BLOCK POSTION TO BEGINNING OF SQUARE
SET DEPTH GROUP NUMBER TO ZERO
1312
1314
SELECT FIRST/ NEXT DATA DISK STARTING WITH DISK ZERO
1316
SKIP PAST DEPTH * (DISK NUMBER + 1) BLOCKS
1318
EXCHANGE DATA
FIG. 13B
1320
END OF SQUARE ?
N
Y
1322
INCREMENT DATA DISK NUMBER, SET DEPTH GROUP NUMBER TO ZERO
1324
END OF DATA DISKS ?
N
Y
RETURN

FIG. 13A

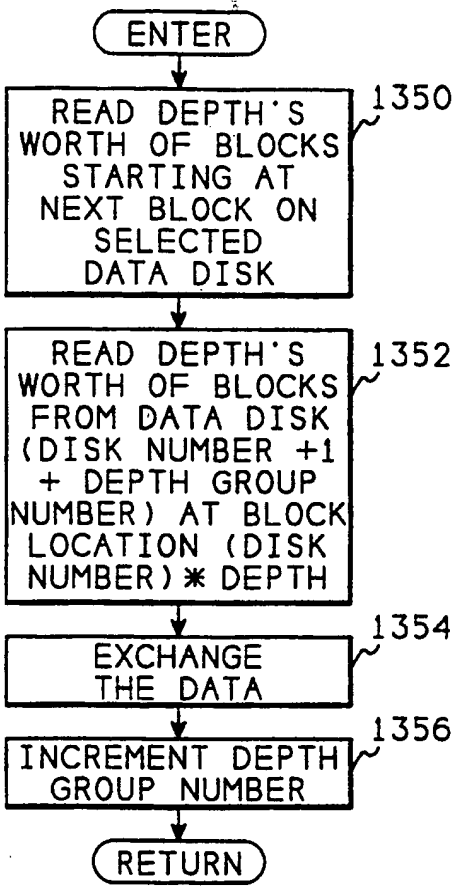

ENTER
READ DEPTH'S WORTH OF BLOCKS STARTING AT NEXT BLOCK ON SELECTED DATA DISK
1350
READ DEPTH'S WORTH OF BLOCKS FROM DATA DISK (DISK NUMBER +1 + DEPTH GROUP NUMBER) AT BLOCK LOCATION (DISK NUMBER) * DEPTH
1352
EXCHANGE THE DATA
1354
INCREMENT DEPTH GROUP NUMBER
1356
RETURN

FIG. 13B

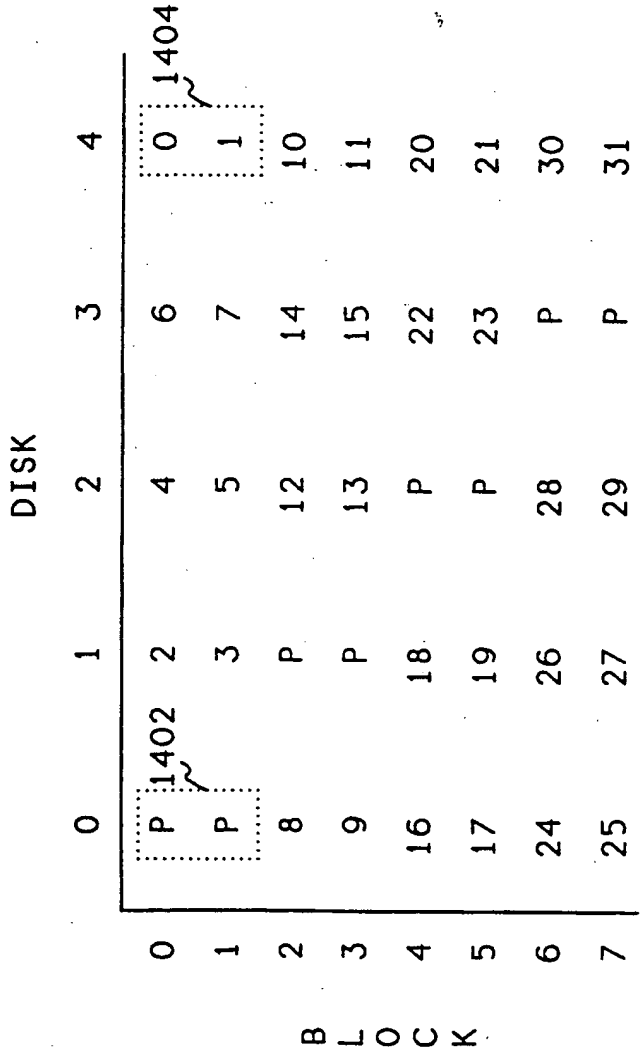

DISK
0 1 2 3 4
BLOCK
0 P 2 4 6 0
1 P 3 5 7 1
2 8 P 12 14 10
3 9 P 13 15 11
4 16 18 P 22 20
5 17 19 P 23 21
6 24 26 28 P 30
7 25 27 29 P 31
1402
1404

FIG. 14

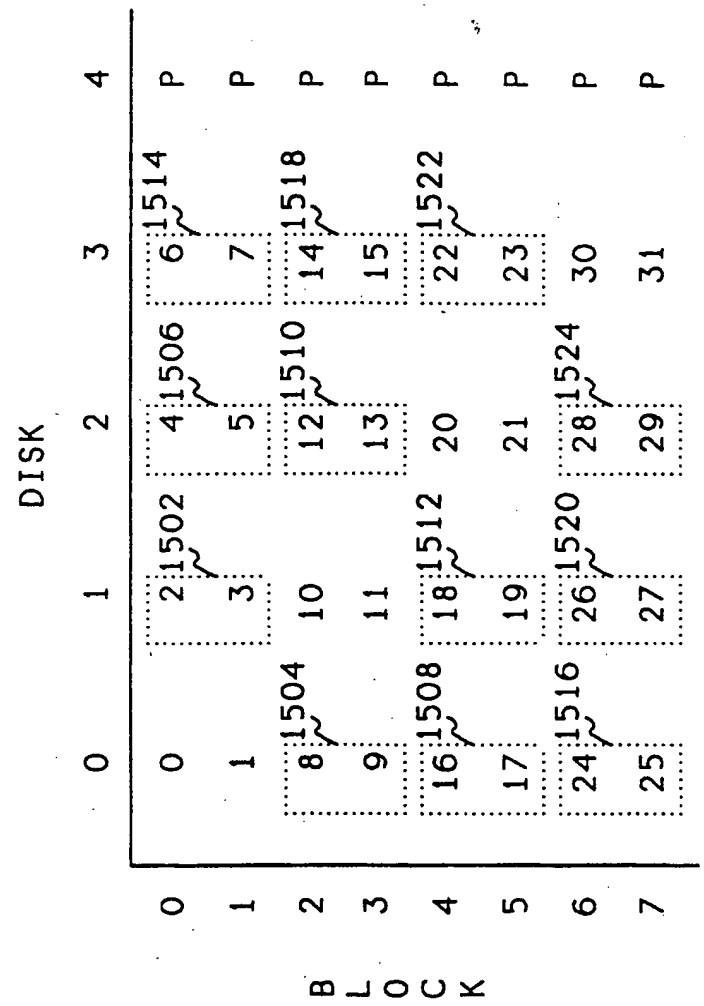

DISK
0
1
2
3
4
BLOCK
0
1
2
3
4
5
6
7
0
1
8
9
16
17
24
25
2
3
10
11
18
19
26
27
4
5
12
13
20
21
28
29
6
7
14
15
22
23
30
31
P
1502
1504
1506
1508
1510
1512
1514
1516
1518
1520
1522
1524

FIG. 15


DISK
0
1
2
3
4
BLOCK
0 0 8 16 24 P
1 1 9 17 25 P
2 2 10 18 26 P
3 3 11 19 27 P
4 4 12 20 28 P
5 5 13 21 29 P
6 6 14 22 30 P
7 7 15 23 31 P

FIG. 16

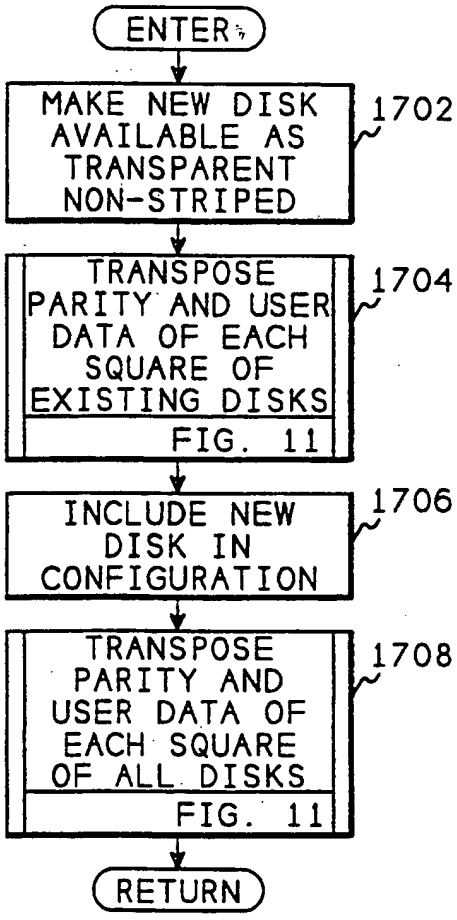

ENTER
MAKE NEW DISK AVAILABLE AS TRANSPARENT NON-STRIPED
1702
TRANSPOSE PARITY AND USER DATA OF EACH SQUARE OF EXISTING DISKS
FIG. 11
1704
INCLUDE NEW DISK IN CONFIGURATION
1706
TRANSPOSE PARITY AND USER DATA OF EACH SQUARE OF ALL DISKS
FIG. 11
1708
RETURN

FIG. 17

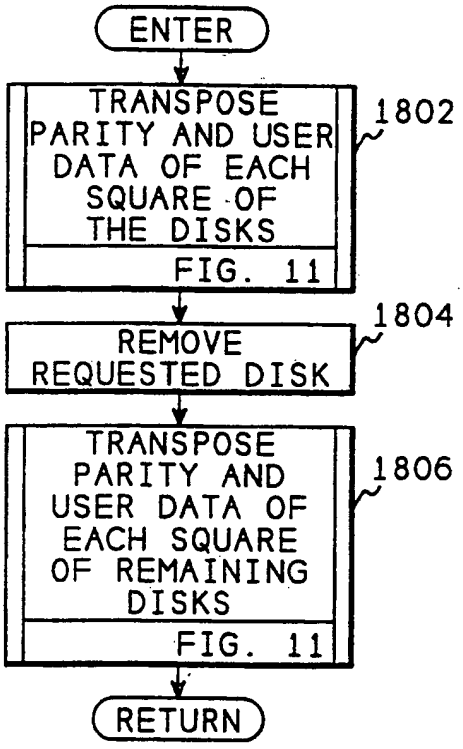

ENTER
TRANSPOSE PARITY AND USER DATA OF EACH SQUARE OF THE DISKS
FIG. 11
1802
REMOVE REQUESTED DISK
1804
TRANSPOSE PARITY AND USER DATA OF EACH SQUARE OF REMAINING DISKS
FIG. 11
1806
RETURN

FIG. 18

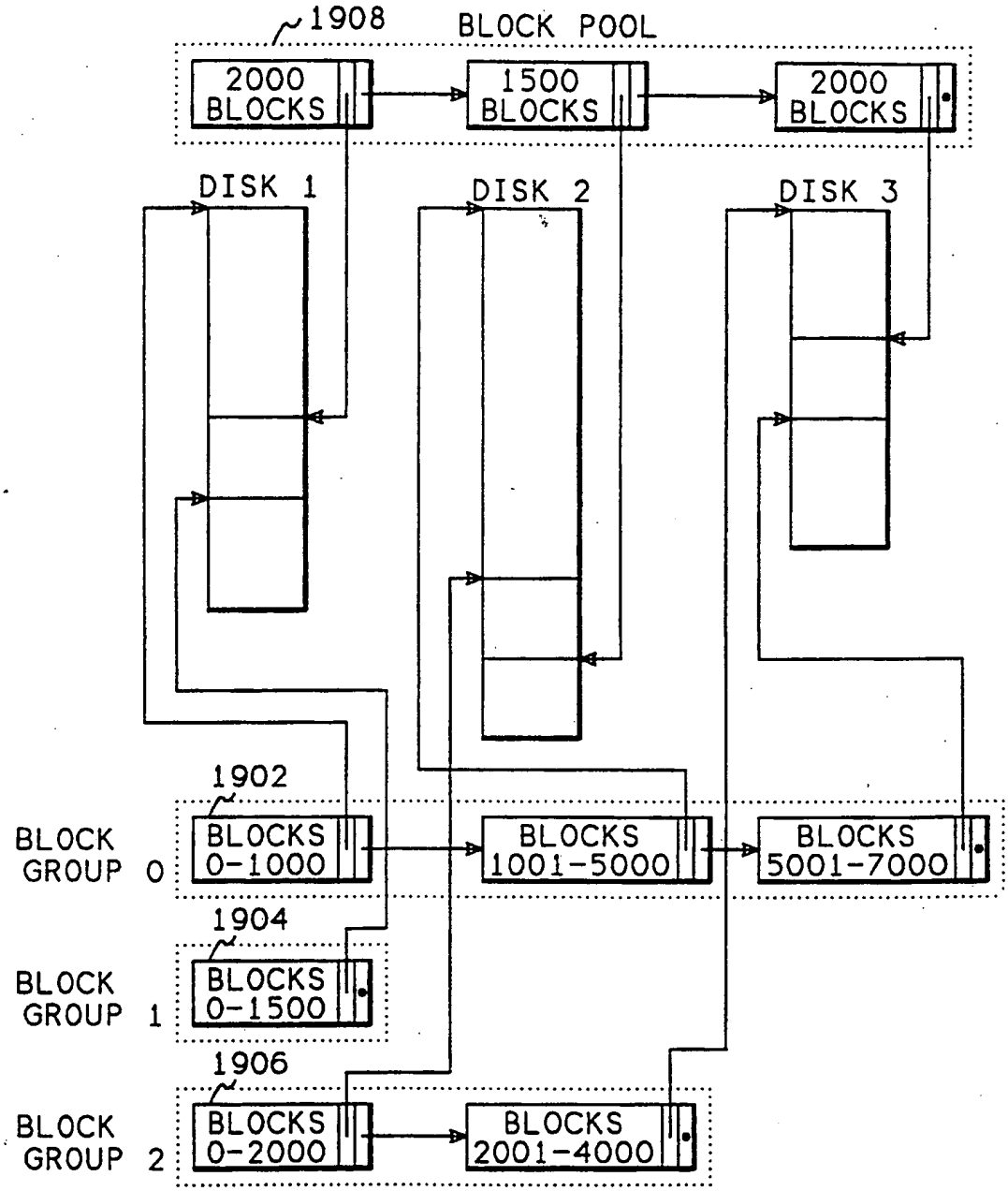

1908
BLOCK POOL
2000 BLOCKS
1500 BLOCKS
2000 BLOCKS
DISK 1
DISK 2
DISK 3
1902
BLOCK GROUP 0
BLOCKS 0-1000
BLOCKS 1001-5000
BLOCKS 5001-7000
1904
BLOCK GROUP 1
BLOCKS 0-1500
1906
BLOCK GROUP 2
BLOCKS 0-2000
BLOCKS 2001-4000

FIG. 19

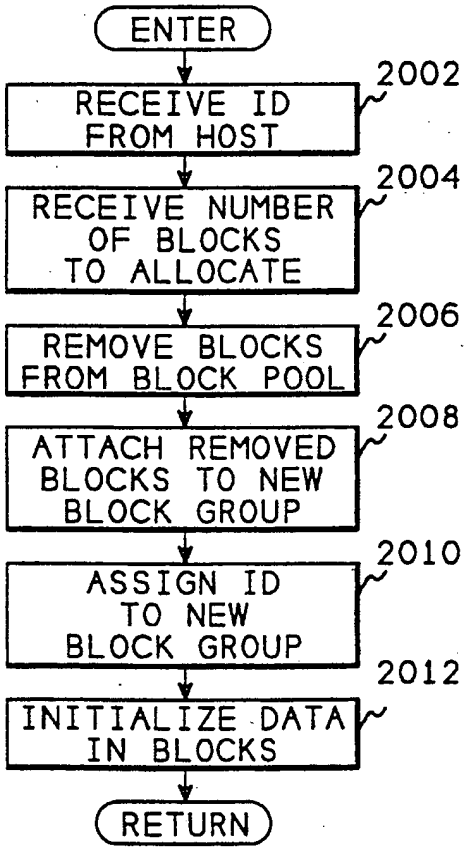

ENTER
RECEIVE ID FROM HOST
2002
RECEIVE NUMBER OF BLOCKS TO ALLOCATE
2004
REMOVE BLOCKS FROM BLOCK POOL
2006
ATTACH REMOVED BLOCKS TO NEW BLOCK GROUP
2008
ASSIGN ID TO NEW BLOCK GROUP
2010
INITIALIZE DATA IN BLOCKS
2012
RETURN

FIG. 20

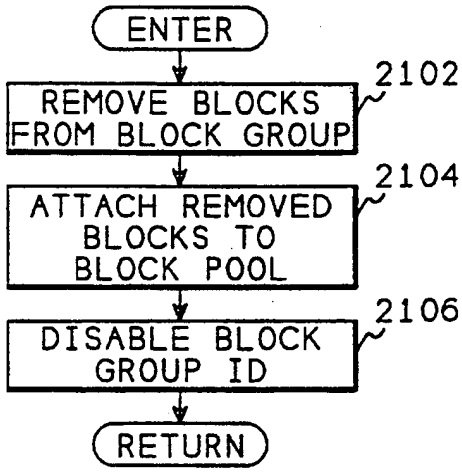

ENTER
REMOVE BLOCKS FROM BLOCK GROUP
2102
ATTACH REMOVED BLOCKS TO BLOCK POOL
2104
DISABLE BLOCK GROUP ID
2106
RETURN

FIG. 21

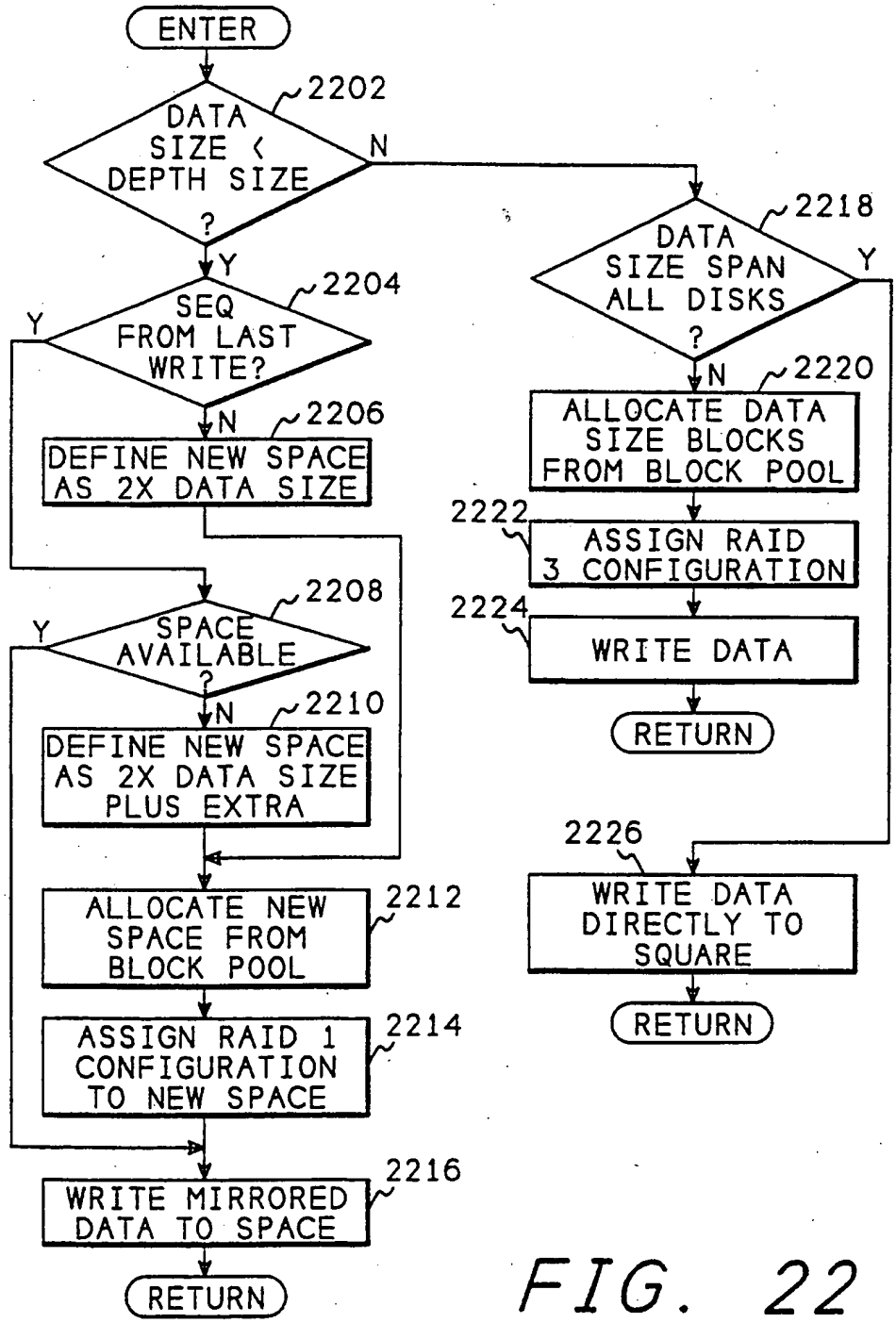

ENTER
2202
DATA SIZE < DEPTH SIZE ?
N
Y
2204
SEQ FROM LAST WRITE?
Y
N
2206
DEFINE NEW SPACE AS 2X DATA SIZE
2208
SPACE AVAILABLE ?
Y
N
2210
DEFINE NEW SPACE AS 2X DATA SIZE PLUS EXTRA
ALLOCATE NEW SPACE FROM BLOCK POOL
2212
ASSIGN RAID 1 CONFIGURATION TO NEW SPACE
2214
WRITE MIRRORED DATA TO SPACE
2216
RETURN
2218
DATA SIZE SPAN ALL DISKS ?
Y
N
2220
ALLOCATE DATA SIZE BLOCKS FROM BLOCK POOL
2222
ASSIGN RAID 3 CONFIGURATION
2224
WRITE DATA
RETURN
2226
WRITE DATA DIRECTLY TO SQUARE
RETURN

FIG. 22

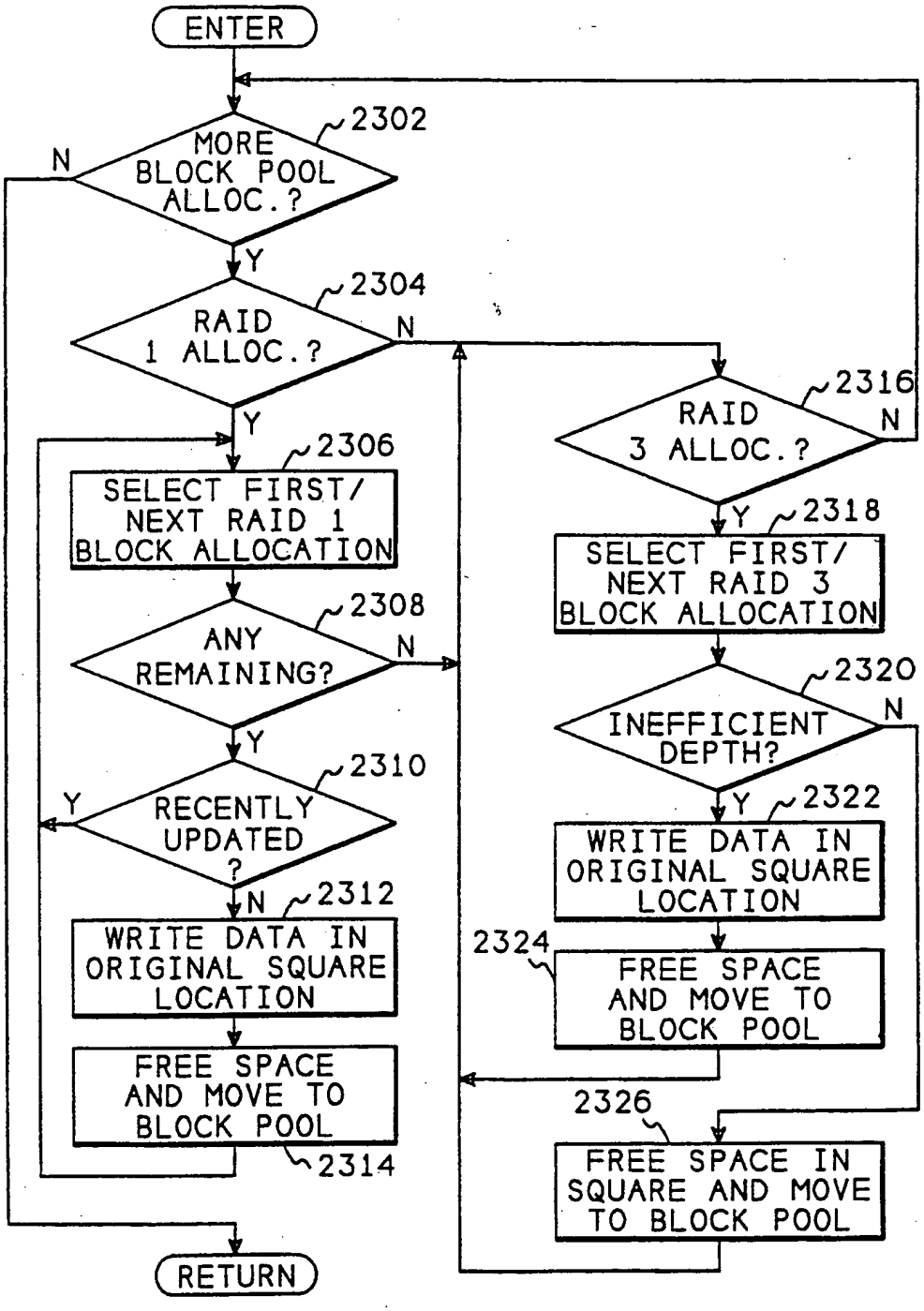

ENTER
2302
MORE BLOCK POOL ALLOC.?
N
Y
2304
RAID 1 ALLOC.?
N
Y
2306
SELECT FIRST/ NEXT RAID 1 BLOCK ALLOCATION
2308
ANY REMAINING?
N
Y
2310
RECENTLY UPDATED ?
Y
N
2312
WRITE DATA IN ORIGINAL SQUARE LOCATION
FREE SPACE AND MOVE TO BLOCK POOL
2314
RETURN
2316
RAID 3 ALLOC.?
N
Y
2318
SELECT FIRST/ NEXT RAID 3 BLOCK ALLOCATION
2320
INEFFICIENT DEPTH?
N
Y
2322
WRITE DATA IN ORIGINAL SQUARE LOCATION
2324
FREE SPACE AND MOVE TO BLOCK POOL
2326
FREE SPACE IN SQUARE AND MOVE TO BLOCK POOL

FIG. 23

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US96/13238

A. CLASSIFICATION OF SUBJECT MATTER

IPC(6) :G06F 12/00, 13/00
US CL :395/439, 441, 497.01, 497.02, 283
According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)

U.S. : 395/439, 441, 497.01, 497.02, 283

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)

APS, DIALOG

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	Business Wire, p03061112, "HP Announces HP AutoRAID Technology; High-availability Storage Technology Automates RAID Level Selection, Provides Substantial Performance Improvement." Issued March 6, 1995, full article.	1-10
X	Business Wire, P7171170, "HP Announces First HP AutoRAID Product; New Disk Array Scheduled for OEM Distribution in Fall with Evaluation Units Now Shipping.", July 17, 1995, full article.	1-10
A,P	US, A, 5,479,653 (JONES) 26 DECEMBER 1995	1-10
A,P	US, A, 5,517,632 (MATSUMOTO ET AL) 14 MAY 1996	1-10

[X] Further documents are listed in the continuation of Box C. [ ] See patent family annex.

*	Special categories of cited documents:	"T"	later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention
"A"	document defining the general state of the art which is not considered to be part of particular relevance	"X"	document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone
"E"	earlier document published on or after the international filing date	"Y"	document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art
"L"	document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)	"&"	document member of the same patent family
"O"	document referring to an oral disclosure, use, exhibition or other means		
"P"	document published prior to the international filing date but later than the priority date claimed		

Date of the actual completion of the international search
29 SEPTEMBER 1996

Date of mailing of the international search report
25 OCT 1996

Name and mailing address of the ISA/US
Commissioner of Patents and Trademarks
Box PCT
Washington, D.C. 20231
Facsimile No. (703) 305-3230

Authorized officer
FRANK J. ASTA
Telephone No. (703) 305-3817

Form PCT/ISA/210 (second sheet)(July 1992)*

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US96/13238

C (Continuation). DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
A,P	US, A, 5,519,844 (STALLMO) 21 MAY 1996	1-10
A,P	US, A, 5,524,204 (VERDOON, JR.) 04 JUNE 1996	1-10